on the transition system $\Upsilon_{\mathcal{K}}$ leads to a state $\mathcal{M}_n$ such that the tuple $\vec{t} = b$ is in the set $\mathsf{ANS}(q, \mathcal{K}_n = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_n))$ for q the query

$$q[x] = \ \mathsf{S3{::}Bucket}(x) \ \wedge \ \textsc{Must}\ \big(\exists y, z.\ \mathsf{bucketSSEncryption}(x, y) \wedge \\ \mathsf{bucketKey}(y, z) \ \wedge \ \mathsf{enableKeyRotation}(z, true)\big)$$

It is easy to see that the following three sequences of grounded actions are valid plans from $\mathcal{K}$ to (b, q):

$$\pi_1 = \mathsf{enableKeyRotation}(k)$$
$$\pi_2 = \mathsf{createKey}(k_1) \cdot \mathsf{enableKeyRotation}(k_1) \cdot \mathsf{putBucketEncryption}(b, k_1)$$
$$\pi_3 = \mathsf{deleteBucketEncryption}(b, k) \cdot \mathsf{createKey}(k_1) \cdot \mathsf{enableKeyRotation}(k_1) \cdot \\ \mathsf{putBucketEncryption}(b, k_1)$$

If, for example, a bucket was only allowed to have one encryption (by means of a functional axiom in $\mathcal{S}$), then π_2 would not be a valid plan, as it would generate an inconsistent run leading to a state $\mathcal{M}_i$ that is not open-consistent w.r.t. $\mathcal{S}$.

Lemma 3. *The plan existence problem for a finite transition system $\Upsilon_{\mathcal{K}}$ generated by a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base $\mathcal{K}$ and a set of actions* Act*, over a finite domain of objects $\mathcal{D}$, reduces to graph reachability over a graph whose number of states is at most exponential in the size of $\mathcal{D}$.*

The Plan Synthesis Problem. We now focus on the problem of finding plans that satisfy a given condition. As discussed in the previous paragraph, we are mostly driven by query answering; in particular, by conditions corresponding to a tuple (of objects from our starting deployment configuration) satisfying a given requirement expressed as a Must/May query. Clearly, this problem is meaningful in our application of interest because it corresponds to finding a set of potential sequences of changes that would allow one to reach a configuration satisfying (resp., not satisfying) one, or more, security mitigations (resp., vulnerabilities). We concentrate on DL-Lite$^{\mathcal{F}}$ core-closed knowledge bases and their generated finite transition systems, where potential fresh objects are drawn from a fixed set $\mathcal{D}$. We are interested in sequences of grounded actions that are minimal and ignore sequences that extend these. We sometimes call such minimal sequences *simple plans.* A plan π from an initial core-closed knowledge base $\mathcal{K}$ to a goal condition b is minimal (or simple) *iff* there does not exist a plan π' (from the same initial $\mathcal{K}$ to the same goal condition b) s.t. $\pi = \pi' \cdot \sigma$, for σ a non-empty suffix of grounded actions.

In Algorithm 1, we present a depth-first search algorithm that, starting from $\mathcal{K}$, searches for all simple plans that achieve a given target query membership condition. The transition system $\Upsilon_{\mathcal{K}}$ is computed, and stored, on the fly in the $\mathsf{Successors}$ sub-procedure and the graph is explored in a depth-first search traversal fashion.

Algorithm 1: $\mathsf{FindPlans}(\mathcal{K}, \mathcal{D}, \mathsf{Act}, \langle \vec{t}, q\rangle)$

```
Inputs : A ccKB $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_0)$, a domain $\mathcal{D}$, a set of actions Act
         and a pair $\langle \vec{t}, q\rangle$ of an answer tuple and a MUST/MAY query
Output: A possibly empty set $\Pi$ of consistent simple plans
1  def FindPlans ( $\mathcal{K}, \mathcal{D}, \mathsf{Act}, \langle \vec{t}, q\rangle$ ):
2      $\Pi := \emptyset$;
3      $S := \bot$;
4      $\mathsf{AllPlanSearch}(\mathcal{M}_0, \epsilon, \emptyset, \mathcal{K}, \mathcal{D}, \mathsf{Act}, \langle \vec{t}, q\rangle)$ ;
5      return $\Pi$;
6  def AllPlanSearch ( $\mathcal{M}, \pi, V, \mathcal{K}, \mathcal{D}, \mathsf{Act}, \langle \vec{t}, q\rangle$ ):
7      if $\mathcal{M} \in V$ then
8          return;
9      if $\vec{t} \in \mathsf{ANS}(q, \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}\rangle)$ then
10         $\Pi := \Pi \cup \{\pi\}$;
11         return;
12     $Q := \emptyset$;
13     foreach $\left\langle \gamma\vec{\theta}, \mathcal{M}'\right\rangle \in \mathsf{Successors}(\mathcal{M}, \mathsf{Act}, \mathcal{D})$ do
14         $Q.push(\left\langle \gamma\vec{\theta}, \mathcal{M}'\right\rangle)$;
15     $V := V \cup \{\mathcal{M}\}$;
16     while $Q \neq \emptyset$ do
17         $\left\langle \gamma\vec{\theta}, \mathcal{M}'\right\rangle = Q.pop()$;
18         $\mathsf{AllPlanSearch}(\mathcal{M}', \pi \cdot \gamma\vec{\theta}, V, \mathcal{K}, \mathcal{D}, \mathsf{Act}, \langle \vec{t}, q\rangle)$;
19     $V := V \setminus \{\mathcal{M}\}$;
20     return;
21 def Successors ($\mathcal{M}, \mathsf{Act}, \mathcal{D}$):
22     if $S[\mathcal{M}]$ is defined then
23         return $S[\mathcal{M}]$;
24     $N := \emptyset$;
25     foreach $\gamma \in \mathsf{Act}$, $\vec{\theta} \in \mathcal{D}^{ar(\gamma)}$ do
26         $\mathcal{M}' := T_{\gamma\vec{\theta}}(\mathcal{M})$;
27         if $\mathcal{M}'$ is fully satisfiable then
28             $N := N \cup \{\left\langle \gamma\vec{\theta}, \mathcal{M}'\right\rangle\}$
29     $S[\mathcal{M}] := N$;
30     return $N$;
```

We note that the condition $\vec{t} \in \mathsf{ANS}(q, \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}\rangle)$ (line 9) could be replaced by any other query satisfiability condition and that one could easily rewrite the algorithm to be parameterized by a more general boolean goal. For example, the condition that a given tuple $\vec{t}$ is *not* an answer to a query q over the analyzed state, with the query q representing an undesired configuration, or a boolean formula over multiple query membership assertions. We also note that Algorithm 1 could be simplified to return only one simple plan, if a plan exists, or NULL, if a plan does not exist, thus solving the so-called *plan generation problem*. We refer the reader to the full version of this paper [16] containing the plan generation algorithm (full version, Appendix A.1) and the proofs of Theorem 2 and 3 below (full version, Appendices A.2 and A3, respectively).

Theorem 2 (Minimal Plan Synthesis Correctness). *Let $\mathcal{K}$ be a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base, $\mathcal{D}$ be a fixed finite domain,* Act *be a set of ungrounded action labels, and $\langle \vec{t}, q \rangle$ be a goal. Then a plan π is returned by the algorithm* FindPlans$(\mathcal{K}, \mathcal{D}, \mathsf{Act}, \langle \vec{t}, q \rangle)$ *if and only if π is a minimal plan from $\mathcal{K}$ to $\langle \vec{t}, q \rangle$.*

Theorem 3 (Minimal Plan Synthesis Complexity). *The* FindPlans *algorithm runs in polynomial time in the size of $\mathcal{M}$ and exponential time in size of $\mathcal{D}$.*

7 Related Work

The syntax of the action language that we presented in this paper is similar to that of [1,12,13]. Differently from their work, we disallow complex action effects to be nested inside conditional statements, and we define basic action effects that consist purely in the addition and deletion of concept and role $\mathcal{M}$-assertions. Thus, our actions are much less general than those used in their framework. The semantics of their action language is defined in terms of changes applied to instances, and the action effects are captured and encoded through a variant of $\mathcal{ALCHOIQ}$ called $\mathcal{ALCHOIQ}_{br}$. In our work, instead, the execution of an action updates a portion of the core-closed knowledge base $\mathcal{K}$—the core $\mathcal{M}$, which is interpreted under a close-world assumption and can be seen as a partial assignment for the interpretations that are models of $\mathcal{K}$. Since we directly manipulate $\mathcal{M}$, the semantics of our actions is more similar to that of [21] and, in general, to ABox updates [22,23]. Like the frameworks introduced in [9–11,20], our actions are parameterized and when combined with a core-closed knowledge base generate a transition system. In [11], the authors focus on a variant of *Knowledge and Action Bases* [21] called *Explicit-Input KABs* (eKABs); in particular, on finite and on state-bounded eKABs, for which planning existence is decidable. Our generated transition systems are an adaptation of the work done in *Description Logic based Dynamic Systems*, *KABs*, and *eKABs* to our setting of core-closed knowledge bases. In [24], the authors address decidability of the plan existence problem for logics that are subset of $\mathcal{ALCOI}$. Their action language is similar to the one presented in this paper; including pre-conditions, in the form of a set of ABox assertions, post-conditions, in the form of basic addition or removal of assertions, concatenation, and input parameters. In [11], the plan synthesis problem is discussed also for lightweight description logics. Relying on the FOL-reducibility of DL-Lite$^{\mathcal{A}}$, it is shown that plan synthesis over DL-Lite$^{\mathcal{A}}$ can be compiled into an ADL planning problem [25]. This does not seem possible in our case, as not all necessary tests over core-closed knowledge bases are known to be FOL-reducible. In [10] and [9], the authors concentrate on verifying and synthesizing temporal properties expressed in a variant of μ-calculus over description logic based dynamic systems, both problems are relevant in our application scenario and we will consider them in future works.

8 Conclusion

We focused on the problem of analyzing cloud infrastructure encoded as description logic knowledge bases combining complete and incomplete information. From a practical standpoint, we concentrated on formalizing and foreseeing the impact of potential changes pre-deployment. We introduced an action language to encode mutating actions, whose semantics is given in terms of changes induced to the complete portion of the knowledge base. We defined the static verification problem as the problem of deciding whether the execution of an action, no matter the specific parameters passed, always preserves a set of properties of the knowledge base. We characterized the complexity of the problem and provided procedural steps to solve it. We then focused on three formulations of the classical AI planning problem: namely, plan existence, generation, and synthesis. In our setting, the planning problem is formulated with respect to the transition system arising from the combination of a core-closed knowledge base and a set of actions; goals are given in terms of one, or more, Must/May conjunctive query membership assertion; and plans of interest are simple sequences of parameterized actions.

Acknowledgments. This work is supported by the ERC Consolidator grant D-SynMA (No. 772459).

References

1. Ahmetaj, S., Calvanese, D., Ortiz, M., Simkus, M.: Managing change in graph-structured data using description logics. ACM Trans. Comput. Log. **18**(4), 27:1–27:35 (2017)
2. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-lite family and relations. J. Artif. Intell. Res. **36**, 1–69 (2009)
3. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press, Cambridge (2017)
4. Backes, J., et al.: Reachability analysis for AWS-based networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11562, pp. 231–241. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25543-5_14
5. Backes, J., et al.: Stratified abstraction of access control policies. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 165–176. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_9

6. Backes, J., et al.: Semantic-based automated reasoning for AWS access policies using SMT. In: Bjørner, N., Gurfinkel, A. (eds.) 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, 30 October–2 November 2018, pp. 1–9. IEEE (2018). https://doi.org/10.23919/FMCAD.2018.8602994
7. Bouchet, M., et al.: Block public access: trust safety verification of access control policies. In: Devanbu, P., Cohen, M.B., Zimmermann, T. (eds.) ESEC/FSE 2020: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, 8–13 November 2020, pp. 281–291. ACM (2020). https://doi.org/10.1145/3368089.3409728
8. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: EQL-lite: effective first-order query processing in description logics. In: Veloso, M.M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, Hyderabad, India, 6–12 January 2007, pp. 274–279 (2007). http://ijcai.org/Proceedings/07/Papers/042.pdf
9. Calvanese, D., De Giacomo, G., Montali, M., Patrizi, F.: Verification and synthesis in description logic based dynamic systems. In: Faber, W., Lembo, D. (eds.) RR 2013. LNCS, vol. 7994, pp. 50–64. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39666-3_5
10. Calvanese, D., Montali, M., Patrizi, F., Giacomo, G.D.: Description logic based dynamic systems: modeling, verification, and synthesis. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, 25–31 July 2015, pp. 4247–4253. AAAI Press (2015). http://ijcai.org/Abstract/15/604
11. Calvanese, D., Montali, M., Patrizi, F., Stawowy, M.: Plan synthesis for knowledge and action bases. In: Kambhampati, S. (ed.) Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016, pp. 1022–1029. IJCAI/AAAI Press (2016). http://www.ijcai.org/Abstract/16/149
12. Calvanese, D., Ortiz, M., Simkus, M.: Evolving graph databases under description logic constraints. In: Eiter, T., Glimm, B., Kazakov, Y., Krötzsch, M. (eds.) Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, 23–26 July 2013. CEUR Workshop Proceedings, vol. 1014, pp. 120–131. CEUR-WS.org (2013). http://ceur-ws.org/Vol-1014/paper_82.pdf
13. Calvanese, D., Ortiz, M., Simkus, M.: Verification of evolving graph-structured data under expressive path constraints. In: Martens, W., Zeume, T. (eds.) 19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, 15–18 March 2016. LIPIcs, vol. 48, pp. 15:1–15:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.ICDT.2016.15
14. Cauli, C., Li, M., Piterman, N., Tkachuk, O.: Pre-deployment security assessment for cloud services through semantic reasoning. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 767–780. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_36
15. Cauli, C., Ortiz, M., Piterman, N.: Closed- and open-world reasoning in dl-lite for cloud infrastructure security. In: Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Hanoi, Vietnam (2021)
16. Cauli, C., Ortiz, M., Piterman, N.: Actions over core-closed knowledge bases (2022). https://doi.org/10.48550/ARXIV.2202.12592. https://arxiv.org/abs/2202.12592
17. Chapman, D.: Planning for conjunctive goals. Artif. Intell. **32**(3), 333–377 (1987). https://doi.org/10.1016/0004-3702(87)90092-0

18. Cook, B.: Formal reasoning about the security of amazon web services. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 38–47. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_3
19. Erol, K., Nau, D.S., Subrahmanian, V.S.: Complexity, decidability and undecidability results for domain-independent planning. Artif. Intell. **76**(1–2), 75–88 (1995). https://doi.org/10.1016/0004-3702(94)00080-K
20. Giacomo, G.D., Masellis, R.D., Rosati, R.: Verification of conjunctive artifact-centric services. Int. J. Cooperative Inf. Syst. **21**(2), 111–140 (2012). https://doi.org/10.1142/S0218843012500025
21. Hariri, B.B., Calvanese, D., Montali, M., Giacomo, G.D., Masellis, R.D., Felli, P.: Description logic knowledge and action bases. J. Artif. Intell. Res. **46**, 651–686 (2013)
22. Kharlamov, E., Zheleznyakov, D., Calvanese, D.: Capturing model-based ontology evolution at the instance level: the case of dl-lite. J. Comput. Syst. Sci. **79**(6), 835–872 (2013). https://doi.org/10.1016/j.jcss.2013.01.006
23. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Foundations of instance level updates in expressive description logics. Artif. Intell. **175**(18), 2170–2197 (2011). https://doi.org/10.1016/j.artint.2011.08.003
24. Milicic, M.: Planning in action formalisms based on DLS: first results. In: Calvanese, D., et al. (eds.) Proceedings of the 2007 International Workshop on Description Logics (DL2007), Brixen-Bressanone, near Bozen-Bolzano, Italy, 8–10 June 2007. CEUR Workshop Proceedings, vol. 250. CEUR-WS.org (2007). http://ceur-ws.org/Vol-250/paper_59.pdf
25. Pednault, E.P.D.: ADL and the state-transition model of action. J. Logic Comput. **4**(5), 467–512 (1994). https://doi.org/10.1093/logcom/4.5.467
26. Tobies, S.: A NExpTime-complete description logic strictly contained in C^2. In: Flum, J., Rodriguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 292–306. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48168-0_21

GK: Implementing Full First Order Default Logic for Commonsense Reasoning (System Description)

Tanel Tammet[1](✉), Dirk Draheim[2], and Priit Järv[1]

[1] Applied Artificial Intelligence Group, Tallinn University of Technology, Tallinn, Estonia
{tanel.tammet,priit.jarv1}@taltech.ee

[2] Information Systems Group, Tallinn University of Technology, Tallinn, Estonia
dirk.draheim@taltech.ee

Abstract. Our goal is to develop a logic-based component for hybrid – machine learning plus logic – commonsense question answering systems. The paper presents an implementation GK of default logic for handling rules with exceptions in unrestricted first order knowledge bases. GK is built on top of our existing automated reasoning system with confidence calculation capabilities. To overcome the problem of undecidability of checking potential exceptions, GK performs delayed recursive checks with diminishing time limits. These are combined with the taxonomy-based priorities for defaults and numerical confidences.

1 Introduction

The problem of handling uncertainty is one of the critical issues when considering the use of logic for automating commonsense reasoning. Most of the facts and rules people use in their daily lives are uncertain. There are many types of uncertainty, like fuzziness (is a person somewhat tall or very tall), confidence (how certain does some fact seem) and exceptions (birds can typically fly, but penguins, ostriches etc., can not). Some of these uncertainties, like fuzziness and confidence, can be represented numerically, while others, like rules with exceptions, are discrete. In [18] we present the design and implementation of the CONFER framework for extending existing automated reasoning systems with confidence calculation capabilities. In the current paper we present the implementation called GK for default logic [13], built by further extending the CONFER implementation. Importantly, we design a novel practical framework for implementing default logic for the full, undecidable first order logic on the basis of a conventional resolution prover.

1.1 Default Logic

Default logic was introduced in 1980 by R. Reiter [13] to model one aspect of common-sense reasoning: rules with exceptions. It has remained one of the most

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 300–309, 2022.
https://doi.org/10.1007/978-3-031-10769-6_18

well-known logic-based mechanisms devoted to this goal, with the *circumscription* by J. McCarthy and the *autoepistemic logic* being the early alternatives. Several similar systems have been proposed later, like defeasible logic [11].

Default logic [13] extends classical logic with default rules of the form

$$\frac{\alpha(x) : \beta_1(x), ...\beta_n(x)}{\gamma(x)}$$

where a *precondition* $\alpha(\mathrm{x})$, *justifications* $\beta_1(x), ...\beta_n(x)$ and a *consequent* $\gamma(x)$ are first order predicate calculus formulas whose free variables are among $x = x_1, ..., x_m$. For every tuple of individuals $t = t_1, ..., t_n$, if the precondition $\alpha(t)$ is derivable and none of the *negated* justifications $\neg\beta(t)$ are derivable from a given knowledge base KB, then the consequent $\gamma(t)$ can be derived from KB. Differently from classical and most other logics, default logic is *non-monotonic*: adding new assumptions can make some previously derivable formulas non-derivable.

As investigated in [7], the interpretation of quantifiers in default rules can lead to several versions of default logic. We follow the original interpretation of Reiter in [13] which requires the use of Skolemization in a specific manner over default rules. For example, a default rule: $\exists x P(x) \vdash \exists x P(x)$ should be interpreted as $: P(c) \vdash P(c)$, where c is a Skolem constant.

Consider a typical example for default logic: birds can normally fly, but penguins cannot fly. The classical logic part

$$penguin(p)\ \&\ bird(b)\ \&\ \forall x.penguin(x) \Rightarrow bird(x)\ \&\ \forall x.penguin(x) \Rightarrow \neg fly(x).$$

is extended with the default rule $bird(x) : fly(x) \vdash fly(x)$. From here we can derive that an arbitrary bird b can fly, but a penguin p cannot. The default rule cannot be applied to p, since a contradiction is derivable from $fly(p)$. This argument cannot be easily modelled using numerical confidences: the probability of an arbitrary living bird being able to fly is relatively high, while the penguins form a specific subset of birds, for which this probability is zero.

Another well-known example – Nixon's triangle – introduces the problem of multiple extensions and *sceptical* vs *credulous* entailment: the classical facts $republican(nixon)\ \&\ quaker(nixon)$ extended with two mutually excluding default rules $republican(x) : \neg pacifist(x) \vdash \neg pacifist(x)$ and $quaker(x) : pacifist(x) \vdash pacifist(x)$. The credulous entailment allows giving different priorities to the default rules and accepts different sets (*extensions*) of consequences, if there is a way to assign priorities so that all the consequences in an extension can be derived. The sceptical entailment requires that a consequence is present in all extensions. GK follows the latter interpretation, but allows explicit priorities to be assigned to the default rules.

The concept of *priorities* for default rules has been well investigated, with several mechanisms proposed. G. Brewka argues in [4] that "for realistic applications involving default reasoning it is necessary to reason about the priorities of defaults" and introduces an ordering of defaults based on specificity: default rules for a more specific class of objects should take priority over rules for more general classes. For example, since birds (who typically do fly) are physical objects and physical objects typically do not fly, we have contradictory default rules describing the flying capability of arbitrary birds. Since birds are a subset of physical objects, the flying rule of birds should have a higher priority than the non-flying rule of physical objects.

1.2 Undecidability, Grounding and Implementations

Perhaps the most significant problem standing in the way of automating default logic is undecidability of the applicability of rules. Indeed, in order to apply a default rule, we must prove that the justifications do not lead to a contradiction with the rest of the knowledge base KB. For full first order logic this is undecidable. Hence, the standard approach for handling default logic has been creating a large ground instance KB_g of the KB, and then performing decidable propositional reasoning on the KB_g.

Almost all the existing implementations of default logic like DeReS [5], DLV2 [1] or CLINGO [8], with the noteworthy exception of s(CASP) [2], follow the same principle. More generally, the field of *Answer Set Programming* (ASP), see [10], is devoted to this approach. As an exception, the s(CASP) system [2] solves queries without the grounding step and is thus better suited for large domains. It is noteworthy that the s(CASP) system has been used in [9] for automating common sense reasoning for autonomous driving with the help of default rules. However, s(CASP) is a logic programming system, not a universal automated reasoner. For example, when we add a rule `bird(father(X)) :- bird(X)` to the formulation of the above birds example in s(CASP), the search does not terminate, apparently due to the infinitely growing nesting of terms.

While ASP systems are very well suited for specific kinds of problems over a small finite domain, grounding becomes infeasible for large first order knowledge bases (*KB* in the following), in particular when the domain is infinite and nested terms can be derived from the KB. The approach described in this paper accepts the lack of logical omniscience and performs delayed recursive checking of exceptions with diminishing time limits directly on non-grounded clauses, combined with the taxonomy-based priorities for defaults and numerical confidences.

2 Algorithms

Our approach of implementing default rules in GK for first order logic is to delay justification checking until a first-order proof is found and then perform recursively deepening checks with diminishing time limits. Thus, our system first produces a potentially large number of different candidate proofs and then enters a recursive checking phase. The idea of delaying justification checking is already present in the original paper of R. Reiter [13], where he uses linear resolution and delayed checks as the main machinery of his proofs. The results produced by GK thus depend on the time limits and are not stable. Showing specific fixpoint properties of the algorithm is not in the scope of our paper.

A practical question for implementation is the actual representation of default rules and making the rules fit the first-order proof search machinery. To this end we introduce *blocker atoms* which are similar to the justification indexes of Reiter.

In the following we will assume that the underlying first order reasoner uses the resolution method, see [3] for details. The rest of the paper assumes familiarity with the basic concepts, terminology and algorithms of the resolution method.

2.1 Background: Queries and Answers

We assume our system is presented with a question in one of two forms: *(1)* Is the statement Q true? *(2)* Find values V for existentially bound variables in Q so that Q is true. For simplicity's sake we will assume that the statement Q is in the prefix form, i.e., no quantifiers occur in the scope of other logical connectives.

In the second case, it could be that several different value vectors can be assigned to the variables, essentially giving different answers. We also note that an answer could be a disjunction, giving possible options instead of a single definite answer.

A widely used machinery in resolution-based theorem provers for extracting values of existentially bound variables in Q is to use a special *answer predicate*, converting a question statement Q to a formula $\exists X(Q(X)\&\neg answer(X))$ for a tuple of existentially quantified variables X in Q [6]. Whenever a clause is derived which consists of only answer predicates, it is treated as a contradiction (essentially, answer) and the arguments of the answer predicate are returned as the values looked for. A common convention is to call such clauses *answer clauses.* We will require that the proof search does not stop whenever an answer clause is found, but will continue to look for new answer clauses until a predetermined time limit is reached. See [16] for a framework of extracting multiple answers.

We also assume that queries take a general form $(KB\&A) \Rightarrow Q$ where KB is a commonsense knowledge base, A is an optional set of precondition statements for this particular question and Q is a question statement. The whole general query form is negated and converted to clauses, i.e., disjunctions of literals (positive or negative atoms). We will call the clauses stemming from the question statement *question clauses.*

2.2 Blocker Atoms and Justification Checking

Without loss of generality we assume that the precondition and consequent formulas α and γ in default rules are clauses and justifications $\beta_1, ..., \beta_n$ are literals, i.e. positive or negative atoms: $\alpha : \beta_1, ...\beta_n \vdash \gamma$. Complex formulas can be encoded with a new predicate over the free variables of the formula and an equivalence of the new atom with the formula. Recall that Reiter assumes that the default rules are Skolemized.

We encode a default rule as a clause by concatenating into one clause the precondition and consequent clauses $\alpha(x)$ and $\gamma(x)$ and blocker atoms $block(\neg\beta_1)$,

..., *block*$(\neg\beta_n)$ where each justification β_i is either a positive or a negative atom. The negation $\neg$ is used since we prefer to speak about *blockers* and not *justificatons.* For example, the "birds can fly" default rule is represented as a clause

$$\neg\texttt{bird(X)} \vee \texttt{fly(X)} \vee \texttt{block(0,neg(fly(X)))}$$

where `X` is a variable and `neg(fly(X))` encodes the negated justification. The first argument of the blocker (0 above) encodes priority information covered in the next section.

A proof of a question clause is a clause containing only answer atoms and blocker atoms. In the justification checking phase the system attempts to prove each decoded second blocker argument $\neg\beta_i$ in turn: the proof is considered invalid if some of $\neg\beta_i$ can be proved and this checking-proof itself is valid. If we pose a question $\texttt{fly(X)} \Rightarrow \texttt{answer(X)}$ to the system to be proved (see the earlier example), we get two different answers: $\texttt{answer(p)} \vee \texttt{block(neg(fly(p))}$ and $\texttt{answer(b)} \vee \texttt{block(neg(fly(b))}$. Checking the first of these means trying to prove $\neg\texttt{fly(p)}$ which succeeds, hence the first answer is invalid. Checking the second answer we try to prove $\neg\texttt{fly(b)}$ which fails, hence the answer is valid.

Notice that the contents $\neg\beta_i$ of blockers, just like answer clauses, have a role of collecting substitutions during the proof search: this enables us to disregard the order in which the clauses are used, i.e. both top-down, bottom-up and mixed proof search strategies can be used.

Importantly, blockers are used during the subsumption checks similarly to ordinary literals. A clause C_1 with fewer or more general literals than C_2 is hence always preferred to C_2, given that (a) the literals of C_1 subsume C_2, disregarding the priority arguments of blockers, and (b) the priority arguments of corresponding blocker literals in C_1 are equal or stronger than these of C_2. When combined with the uncertainty and inconsistency handling mechanisms of CONFER, the subsumption restrictions of the latter also apply. There are also other differences to ordinary literals. First, we prohibit the application of equality (demodulation or paramodulation) to the contents of blocker atoms during proof search. Second, we discard clauses containing mutually contradictory blockers (assuming the decoding of the second argument) like we would discard ordinary tautologies.

2.3 Priorities, Recursion and Infinite Branches

Default rule priorities are critical for the practical encoding of commonsense knowledge. The usage of priorities in proof search is simple: when checking a blocker with a given priority, it is not allowed to use default rules with a lower priority. We encode priority information as a first argument of the blocker literal, offering several ways to determine priority: either as an integer, a taxonomy class number, a string in a taxonomy or a combination of these with an integer.

For automatically using specificity we employ taxonomy classes: a class has a higher prirority than those above it on the taxonomy branch. We have built a topologically sorted acyclic graph of English words using the WordNet taxonomy along with an efficient algorithm for quick priority checks during proof search. Taxonomy classes are indicated with a special term like $(`61598`). Alternatively one can use an actual English word like $(“`bird`”) which is automatically recognized to be more specific than, say, $(“`object`”). To enable more fine-grained priorities, an integer can be added to the term like $(“`bird`”, `2`) generating a lexicographic order.

The recursive check for the non-provability of blockers can go arbitrarily deep, except for the time limits. Our algorithm allocates N seconds for the whole proof search and spends half of N for looking for different proofs and answers for the query, with the other half split evenly for each answer. Again, the time allocated for checking an answer is split evenly between the blockers in the answer. Each such time snippet is again split between a search for the proof of the blocker, and if found, for recursively checking the validity of this proof. Once the allocated time is below a given threshold (currently one millisecond) the proof is assumed to be not found.

Answers given by the system depend on the amount of time given, the search strategy chosen etc. For example, consider the Nixon triangle presented earlier, with two contradictory default rules. In case the priorities of these rules are equal and we allow defaults with the same priority to be used for checking an answer containing a blocker, the recursive check terminates only because of a time limit, which is unpredictable. Hence, we may sometimes get one answer and sometimes another. In order to increase both stability and efficiency, GK checks the blockers in the search nodes above, and terminates with failure in cases nonterminating loops are detected. Therefore GK always gives a sceptical result to the Nixon triangle: neither $\mathit{pacifist}(\mathit{nixon})$ nor $\neg\mathit{pacifist}(\mathit{nixon})$ is proven.

3 Confidences and Inconsistencies

GK integrates the exception-handling algorithms described in the previous chapter with the algorithms designed for handling inconsistent KB-s and numeric confidences assigned to clauses, previously presented as a CONFER framework in [18]. The framework is built on the resolution method. It calculates the estimates for the confidences of derived clauses, using both (a) the decreasing confidence of a conjunction of clauses as performed by the resolution and paramodulation rule, and (b) the increasing confidence of a disjunction of clauses for cumulating evidence. CONFER handles inconsistent KB-s by requiring the proofs of answers to contain the clauses stemming from the question posed. It performs searches both for the question and its negation and returns the resulting confidence calculated as a difference of the confidences found by these two searches.

The integrated algorithm is more complex than the one we previously described. Whenever the algorithms of the previous chapter speak about “proving”, the system actually performs two independent searches – one for the positive and one for the negated goal – with the confidences calculated for both of these. A blocker is considered to be proved in case the resulting confidence is over a pre-determined configurable threshold, by default 0.5. Blocker proofs must also contain the clause built from the blocker. Thus, the whole search tree for a query consists of two types of interleaved layers: positive/negative confidence searches and blocker checking searches, the latter type potentially making the tree arbitrarily deep up to the minimal time limit threshold.

4 Implementation and Experiments

The described algorithms are implemented by the first author as a software system GK available at https://logictools.org/gk/. GK is written in C on top of our implementation of the CONFER framework [18] which is built on top of a high-performance resolution prover GKC [17] (see https://github.com/tammet/gkc) for conventional first order logic. Thus GK inherits most of the capabilities and algorithms of GKC.

A tutorial and a set of default logic example problems along with proofs from GK are also available at http://logictools.org/gk. GK is able to quickly solve nontrivial problems built by extending classic default logic examples. It is also able to solve classification problems combining exception and cumulative evidence and problems with dynamic situations using fluents, including planning problems. We have built a very large integrated knowledge base from the Quasimodo [14] and ConceptNet [15] knowledge bases, converting these to default logic plus confidences. GK is able to solve simple problems using this large knowledge base along with the Wordnet taxonomy for specificity: see the referenced web page for examples.

The following small example illustrates the fundamental difference of GK from the existing ASP systems for default logic. The standard penguins and birds example presented above in the ASP syntax is

```
bird(b1).
penguin(p1).
bird(X) :- penguin(X).
flies(X) :- bird(X), not -flies(X).
-flies(X) :- penguin(X).
```

Both GK and the ASP systems clingo 5.4.0, dlv 2.1.1 and s(CASP) 0.21.10.09 give an expected answer to the queries `flies(b1)` and `flies(p1)`. However, when we add the rules

```
bird(father(X)) :- bird(X).
penguin(father(X)) :- penguin(X).
```

none of these ASP systems terminate for these queries, while GK does solve the queries as expected. Notably, as pointed out by the author of s(CASP), this system does terminate for the reformulation of the same problem with the two replacement rules

```
flies(X) :- bird(X), not abs(X).
abs(X) :- penguin(X).
```

while clingo and dlv do not terminate. When we instead add the facts and rules

```
father(b1,b2).
father(p1,p2).
...
father(bN-1,bN).
father(pN-1,pN).

ancestor(X,Y):- father(X,Y).
ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).
```

for a large N, s(CASP) does not terminate and clingo and dlv become slow for `flies(b1)`: ca 8 s for $N = 500$ and ca 1 min for $N = 1000$ on a laptop with a 10-th generation i7 processor. GK solves the same question with $N = 1000$ under half a second and with $N = 100000$ under three seconds: the latter problem size is clearly out of scope of the capabilities of existing ASP systems.

We have previously shown that the confidence handling mechanisms in CONFER may slow down proof search for certain types of problems, but do not have a strong negative effect on very large commonsense CYC [12] problems in the TPTP problem collection. Differently from CONFER, the algorithms for default logic described above do not substantially modify the resolution method implementation of pure first order logic search, thus the performance of these parts of GK are mostly the same as of GKC. The ability to give a correct answer to a query during a given time limit depends on the performance of these components, and not on the overall recursively branching algorithm.

5 Summary and Future Work

We have presented algorithms and an implementation of an automated reasoning system for default logic on the basis of unrestricted first order logic and a resolution method. While there are several systems able to solve default logic or similar nonmonotonic logic problems, these are built on the basis of answer set programming and are normally based on grounding. We are not aware of other full first order logic reasoning systems for default logic, and neither of systems integrating confidences and inconsistency-handling with rules with exceptions.

Future work is planned on three directions: adding features to the solver, proving several useful properties of the algorithms and incorporating the solver into a commonsense reasoning system able to handle nontrivial tasks posed in natural language. The work on incorporating similarity-based reasoning into GK and building a suitable semantic parser for natural language is currently ongoing. We are particularly interested in exploring practical ways to integrate GK with the machine learning techniques for natural language.

References

1. Alviano, M., et al.: The ASP system DLV2. In: Balduccini, M., Janhunen, T. (eds.) LPNMR 2017. LNCS (LNAI), vol. 10377, pp. 215–221. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61660-5_19

2. Arias, J., Carro, M., Salazar, E., Marple, K., Gupta, G.: Constraint answer set programming without grounding. Theor. Pract. Logic Program. **18**(3–4), 337–354 (2018)
3. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, ch. 2, pp. 19–99. Elsevier, Amsterdam (2001)
4. Brewka, G.: Adding priorities and specificity to default logic. In: MacNish, C., Pearce, D., Pereira, L.M. (eds.) JELIA 1994. LNCS, vol. 838, pp. 247–260. Springer, Heidelberg (1994). https://doi.org/10.1007/BFb0021977
5. Cholewinski, P., Marek, V.W., Truszczynski, M.: Default reasoning system deres. KR **96**, 518–528 (1996)
6. Green, C.: Theorem proving as a basis for question-answering systems. Mach. Intell. **4**, 183–205 (1969)
7. Kaminski, M.: A comparative study of open default theories. Artif. Intell. **77**(2), 285–319 (1995)
8. Kaminski, R., Schaub, T., Wanko, P.: A tutorial on hybrid answer set solving with *clingo*. In: Ianni, G. (ed.) Reasoning Web 2017. LNCS, vol. 10370, pp. 167–203. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61033-7_6
9. Kothawade, S., Khandelwal, V., Basu, K., Wang, H., Gupta, G.: Auto-discern: Autonomous driving using common sense reasoning (2021). arXiv preprint. arXiv:2110.13606
10. Lifschitz, V.: Answer Set Programming. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-24658-7
11. Nute, D.: Defeasible Logic, vol. 3. Oxford University Press, Oxford (1994)
12. Ramachandran, D., Reagan, P., Goolsbey, K.: First-orderized researchcyc: expressivity and efficiency in a common-sense ontology. In: AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications, pp. 33–40 (2005)
13. Reiter, R.: A logic for default reasoning. Artif. Intell. **13**(1–2), 81–132 (1980)
14. Romero, J., Razniewski, S., Pal, K., Pan, J.Z., Sakhadeo, A., Weikum, G.: Commonsense properties from query logs and question answering forums. In: Zhu, W. (eds.) Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM'19, pp. 1411–1420. ACM (2019)
15. Speer, R., Chin, J., Havasi, C.: ConceptNet 5.5: An open multilingual graph of general knowledge. In: Singh, S.P., Markovitch, S. (eds.) Proceedings of the 31st AAAI Conference on Artificial Intelligence, pp. 4444–4451. AAAI (2017)
16. Sutcliffe, G., Yerikalapudi, A., Trac, S.: Multiple answer extraction for question answering with automated theorem proving systems. In: Lane, H.C., Guesgen, H.W. (eds.) Proceedings of the 22nd International Florida Artificial Intelligence Research Society Conference, FLAIRS'22. AAAI (2009)
17. Tammet, T.: GKC: a reasoning system for large knowledge bases. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 538–549. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_32
18. Tammet, T., Draheim, D., Järv, P.: Confidences for Commonsense Reasoning. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 507–524. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79876-5_29

Hypergraph-Based Inference Rules for Computing $\mathcal{EL}^+$-Ontology Justifications

Hui Yang(✉), Yue Ma, and Nicole Bidoit

LISN, CNRS, Université Paris-Saclay, Gif-sur-Yvette, France
{yang,ma,nicole.bidoit}@lisn.fr

Abstract. To give concise explanations for a conclusion obtained by reasoning over ontologies, *justifications* have been proposed as minimal subsets of an ontology that entail the given conclusion. Even though computing one justification can be done in polynomial time for tractable Description Logics such as $\mathcal{EL}^+$, computing all justifications is complicated and often challenging for real-world ontologies. In this paper, based on a graph representation of $\mathcal{EL}^+$-ontologies, we propose a new set of *inference rules* (called H-rules) and take advantage of them for providing a new method of computing all justifications for a given conclusion. The advantage of our setting is that most of the time, it reduces the number of *inferences* (generated by H-rules) required to derive a given conclusion. This accelerates the enumeration of justifications relying on these inferences. We validate our approach by running real-world ontology experiments. Our graph-based approach outperforms PULi [14], the state-of-the-art algorithm, in most of cases.

1 Introduction

Ontologies provide structured representations of domain knowledge that are suitable for AI reasoning. They are used in various domains, including medicine, biology, and finance. In the domain of ontologies, one of the interesting topics is to provide explanations of reasoning conclusions. To this end, *justifications* have been proposed to offer users a brief explanation for a given conclusion. Computing justifications has been widely explored for different tasks, for instance for debugging ontologies [1,9,11] and computing ontology modules [6]. Extracting just one justification can be easy for tractable ontologies, such as $\mathcal{EL}^+$ [17]. For instance, we can find one justification by deleting unnecessary axioms one by one. However, there may exist more than one justification for a given conclusion. Computing all such justifications is computationally complex and reveals itself to be a challenging problem [18].

There are mainly two different approaches [17] to compute all justifications for a given conclusion, the *black-box* approach and the *glass-box* approach. The *black-box* approach [11] relies only on a reasoner and, as such, can be

This work is funded by the BPI-France (PSPC AIDA: 2019-PSPC-09).

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 310–328, 2022.
https://doi.org/10.1007/978-3-031-10769-6_19

used for ontologies in any existing Description Logics. For example, a simple (naive) *black-box* approach would check all the subsets of the ontology using an existing reasoner and then filter the subset-minimal ones (i.e., justifications). Many advanced and optimized black-box algorithms have been proposed since 2007 [10]. Meanwhile, the glass-box approaches have achieved better performances over certain specific ontology languages (such as $\mathcal{EL}^+$-ontology) by going deep into the reasoning process. Among them, the class of SAT-based methods [1–3, 14, 16] performs the best. The main idea developed by SAT-based methods is to trace, in a first step, a *complete set of inferences* (*complete set* for short) that contribute to the derivation of a given conclusion, and then, in a second step, to use SAT-tools or resolution to extract all justifications from these inferences. A detailed example is provided in Sect. 4.1.

In the real world, ontologies are always huge. For instance, the SnomedCT ontology contains more than 300,000 axioms. Thus, the traced *complete set* can be large, which could make it challenging to extract the justifications over them. Several techniques could be applied to reduce the size of the traced *complete set*, like the *locality-based modules* [8] and the *goal-directed tracing algorithm* [12]. One of their shared ideas is to identify, for a given conclusion, a particular part of the ontology relevant for the extraction of justifications. For example, the state-of-the-art algorithm, PULi [14], uses a *goal-directed tracing algorithm*. However, even for PULi, a simple ontology $\mathcal{O} = \{A_i \sqsubseteq A_{i+1} \mid 1 \leq i \leq n-1\}$ with the conclusion $A_0 \sqsubseteq A_n$ leads to a *complete set* containing $n-1$ inferences. This set can not be reduced further even with the previously mentioned optimizations. From this observation, we decided to explore a new SAT-based glass-box method to handle such situations better.

Now, let us look carefully at the ontology $\mathcal{O}$ above, and let us regard each A_i as a graph node N_{A_i}. Then we are able to construct, for $\mathcal{O}$, a directed graph whose edges are of the form $N_{A_i} \rightarrow N_{A_{i+1}}$. It turns out that all the justifications for the conclusion $A_0 \sqsubseteq A_n$ are extracted from all the paths from N_{A_0} to N_{A_n}, and here we have only one such path. We can easily extend this idea on $\mathcal{EL}^+$-ontology because most of the $\mathcal{EL}^+$-axioms can be interpreted as direct edges except one case (i.e., $A \equiv B_1 \sqcap \cdots \sqcap B_n$), for which we need a hyperedge (for more details see Definition 3). However, for more expressive ontologies, this translation becomes more complicated. For example, it is hard to map $\mathcal{ALC}$-axioms to edges as those axioms may contain negation or disjunction of concepts.

This example inspired us to explore a hypergraph representation of the ontology and reformulate inferences and justifications. Roughly, our inferences are built from elementary paths of the hypergraph and lead to particular paths called H-paths. Then, computing all the justifications for a given conclusion is made using such H-paths. For the previous ontology $\mathcal{O}$ and the conclusion $A_0 \sqsubseteq A_n$, our *complete set* is reduced to only two inferences (no matter the value of n) corresponding to the unique path from N_{A_0} to N_{A_n}. The source of improvement provided by our method is twofold. On the one hand, it comes from the fact that elementary paths are pre-computed while extracting the inferences and that existing algorithms like depth-first search can efficiently compute such paths. On the other hand, yet as a consequence, decreasing the size of the *complete sets* of inferences leads to smaller inputs for the SAT-based algorithm extracting justifications from the *complete set* (recall here that our method is a SAT-based glass-box method).

The paper is organized as follows. Section 2 introduces preliminary definitions and notions. In Sect. 3, we associate a hypergraph representation to $\mathcal{EL}^+$-ontology and introduce a new set of inference rules, called H-rules, that generate our inferences. In Sect. 4, we develop the algorithm `minH`, which compute justifications based on our inferences. Section 5 shows experimental results and Sect. 6 summarizes our work.

2 Preliminaries

2.1 $\mathcal{EL}^+$-Ontology

Given sets of atomic concepts $\mathsf{N}_C = \{A, B, \cdots\}$ and atomic roles $\mathsf{N}_R = \{r, s, t, \cdots\}$, the set of $\mathcal{EL}^+$concepts C and axioms α are built by the following grammar rules:

$$C ::= \top \mid A \mid C \sqcap C \mid \exists r.C, \quad a ::= C \sqsubseteq C \mid C \equiv C \mid r \sqsubseteq s \mid r_1 \circ \cdots \circ r_n \sqsubseteq s.$$

A $\mathcal{EL}^+$-ontology $\mathcal{O}$ is a finite set of $\mathcal{EL}^+$-axioms. An **interpretation** $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{O}$ consists of a non-empty set $\triangle^{\mathcal{I}}$ and a mapping from atomic concepts $A \in \mathsf{N}_C$ to a subset $A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$ and from roles $r \in \mathsf{N}_R$ to a subset $r^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$. For a concept C built from the grammar rules, we define $C^{\mathcal{I}}$ inductively by: $(\top)^{\mathcal{I}} = \triangle^{\mathcal{I}}, (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(\exists r.C)^{\mathcal{I}} = \{a \in \triangle^{\mathcal{I}} \mid \exists b, (a, b) \in r^{\mathcal{I}}, b \in C^{\mathcal{I}}\}$, $(r \circ s)^{\mathcal{I}} = \{(a, b) \in \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}} \mid \exists c, (a, c) \in r^{\mathcal{I}}, (c, b) \in s^{\mathcal{I}}\}$. An interpretation is a **model** of $\mathcal{O}$ if it is compatible with all axioms in $\mathcal{O}$, i.e., for all $C \sqsubseteq D, C \equiv D, r \sqsubseteq s, r_1 \circ \cdots \circ r_n \sqsubseteq s \in \mathcal{O}$, we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}, C^{\mathcal{I}} = D^{\mathcal{I}}, r^{\mathcal{I}} \subseteq s^{\mathcal{I}}, (r_1 \circ \cdots \circ r_n)^{\mathcal{I}} \subseteq s^{\mathcal{I}}$, respectively. We say $\mathcal{O} \models a$ where α is an axiom iff each model of $\mathcal{O}$ is compatible with α. A concept A is **subsumed** by B w.r.t. $\mathcal{O}$ if $\mathcal{O} \models A \sqsubseteq B$.

Next, we use $A, B, \cdots, G$ (possibly with subscripts) to denote atomic concepts and we use X, Y, Z (possibly with subscripts) to denote atomic concepts $A, \cdots, G$, or complex concepts $\exists r.A, \cdots, \exists r.G$.

We assume that ontologies are normalized. A $\mathcal{EL}^+$-ontology $\mathcal{O}$ is **normalized** if all its axioms are of the form $A \equiv B_1 \sqcap \cdots \sqcap B_m, A \sqsubseteq B_1 \sqcap \cdots \sqcap B_m, A \equiv \exists r.B, A \sqsubseteq \exists r.B, r \sqsubseteq s$, or $r \circ s \sqsubseteq t$, where $A, B, B_i \in \mathsf{N}_C$, and $r, s, t \in \mathsf{N}_R$. Every $\mathcal{EL}^+$-ontology can be normalised in polynomial time by introducing new atomic concepts and atomic roles.

Example 1. *The following set of axioms is a $\mathcal{EL}^+$-ontology:*
$\mathcal{O} = \{\ a_1{:}A \sqsubseteq D,\ a_2{:}D \sqsubseteq \exists r.E,\ a_3{:}E \sqsubseteq F, a_4{:}B \equiv \exists t.F,\ a_5{:}r \sqsubseteq t,\ a_6{:}G \equiv C \sqcap B\ , a_7{:}C \sqsubseteq A\}$.

It is clear that $\mathcal{O} \models A \sqsubseteq \exists r.E$ as for all models $\mathcal{I}$, we have $A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ by the axiom a_1 and $D^{\mathcal{I}} \subseteq (\exists r.E)^{\mathcal{I}}$ by a_2.

Table 1. Inference rules over $\mathcal{EL}^+$-ontology.

$$\mathcal{R}_1 : \frac{A \sqsubseteq A_1, \cdots, A \sqsubseteq A_n, \quad A_1 \sqcap A_2 \sqcap \cdots \sqcap A_n \sqsubseteq B}{A \sqsubseteq B}$$

$$\mathcal{R}_2 : \frac{A \sqsubseteq A_1, \quad A_1 \sqsubseteq \exists r.B}{A \sqsubseteq \exists r.B} \qquad \mathcal{R}_3 : \frac{A \sqsubseteq \exists r.B_1, \quad B_1 \sqsubseteq B_2, \quad \exists r.B_2 \sqsubseteq B}{A \sqsubseteq B}$$

$$\mathcal{R}_4 : \frac{A_0 \sqsubseteq \exists r_1.A_1, \quad \cdots, A_{n-1} \sqsubseteq \exists r_n.A_n, \quad r_1 \circ \cdots \circ r_n \sqsubseteq r}{A_0 \sqsubseteq \exists r.A_n}$$

2.2 Inference, Support and Justification

Given a $\mathcal{EL}^+$-ontology $\mathcal{O}$, a major reasoning task over $\mathcal{O}$ is *classification*, which aims at finding all subsumptions $\mathcal{O} \models A \sqsubseteq B$ for atomic concepts A, B occurring in $\mathcal{O}$. Generally, it can be solved by applying *inferences* recursively over $\mathcal{O}$ [5].

An **inference** ρ is a pair $\langle \rho_{pre}, \rho_{con} \rangle$ whose *premise* set ρ_{pre} consists of $\mathcal{EL}^+$-axioms and *conclusion* ρ_{con} is a single $\mathcal{EL}^+$-axiom. As usual, a sequence of inferences $\rho^1, \cdots, \rho^n$ is a **derivation** of an axiom α from $\mathcal{O}$ if $\rho^n_{con} = \alpha$ and for any $\beta \in \rho^i_{pre}, 1 \leq i \leq n$, we have $\beta \in \mathcal{O}$ or $\beta = \rho^j_{con}$ for some $j < i$.

As usual, **inference rules** are used to generate inferences. For instance, Table 1 [1,5] shows a set of inference rules for $\mathcal{EL}^+$-ontologies. Next, we use $\mathcal{O} \vdash A \sqsubseteq B$ to denote that $A \sqsubseteq B$ is derivable from $\mathcal{O}$ using inferences generated by the rules in Table 1. The set of inference rules in Table 1 is *sound* and *complete* for classification [5], i.e., $\mathcal{O} \models A \sqsubseteq B$ iff $\mathcal{O} \vdash A \sqsubseteq B$ for any $A, B \in \mathsf{N}_C$.

A **support** of $A \sqsubseteq B$ over $\mathcal{O}$ is a sub-ontology $\mathcal{O}' \subseteq \mathcal{O}$ such that $\mathcal{O}' \models A \sqsubseteq B$. The **justifications** for $A \sqsubseteq B$ are subset-minimal supports of $A \sqsubseteq B$. We denote the collection of all justifications for $A \sqsubseteq B$ w.r.t. $\mathcal{O}$ by $J_{\mathcal{O}}(A \sqsubseteq B)$.

We say S is a **complete set** (of inferences) for $A \sqsubseteq B$ if for any justifications $\mathcal{O}'$ of $A \sqsubseteq B$, we can derive $A \sqsubseteq B$ from $\mathcal{O}'$ using only the inferences in S.

Example 2 (Example 1 cont'd). *Before applying inference rules, axioms in $\mathcal{O}$ are preprocessed in order to be compatible with Table 1. For example, a_4 is replaced by $B \sqsubseteq \exists t.F$ and $\exists t.F \sqsubseteq B$. Then, according to the inference rules of Table 1, we may produce the following inferences: $\rho = \langle \{A \sqsubseteq D, D \sqsubseteq \exists r.E\}, A \sqsubseteq \exists r.E \rangle$, $\rho' = \langle \{A \sqsubseteq \exists r.E, r \sqsubseteq t\}, A \sqsubseteq \exists t.E \rangle$ and $\rho'' = \langle \{A \sqsubseteq \exists t.E, E \sqsubseteq F, \exists t.F \sqsubseteq B\}, A \sqsubseteq B \rangle$ generated by rule $\mathcal{R}_2$, $\mathcal{R}_4$ and $\mathcal{R}_3$ respectively. Then $\mathcal{O} \vdash A \sqsubseteq B$ since $A \sqsubseteq B$ is derivable from $\mathcal{O}$ by the sequence ρ, ρ', ρ''.*

Notice that $\mathcal{O}' = \{a_1, a_2, a_3, a_4, a_5\}$ is a support for $A \sqsubseteq B$, and thus, any superset $\mathcal{O}''$ of $\mathcal{O}'$ is a support of $A \sqsubseteq B$. $\mathcal{O}'$ is also one of the justifications for $A \sqsubseteq B$ as for any $\mathcal{O}''' \subset \mathcal{O}'$, we have $\mathcal{O}''' \not\models A \sqsubseteq B$. Moreover, here the three inferences ρ, ρ', ρ'' provide a complete set for $A \sqsubseteq B$.

3 Hypergraph-Based Inference Rules

3.1 H-Inferences

In general, a (directed) hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ and a set of hyperedges $\mathcal{E}$ [4,7]. A hyperedge is of the form $e = (S_1, S_2), S_1, S_2 \subseteq \mathcal{V}$. In this paper, a hypergraph is associated to an ontology as follows:

Definition 3. *For a given $\mathcal{EL}^+$-ontology $\mathcal{O}$, the associated hypergraph is $\mathcal{G}_\mathcal{O} = (\mathcal{V}_\mathcal{O}, \mathcal{E}_\mathcal{O})$ where (i) the set of nodes $\mathcal{V}_\mathcal{O} = \{N_A, N_r, N_{\exists r.A} \mid A \in \mathsf{N}_C, r \in \mathsf{N}_R\}$ and (ii) the set of edges $\mathcal{E}_\mathcal{O}$ is defined by $f(\mathcal{O})$ where f is the multi-valued mapping shown in Fig. 1. Given a hyperedge e of $\mathcal{E}_\mathcal{O}$, the inverse image of e, $f^{-1}(e)$, is defined in the obvious manner. For a set E of hyperedges, $f^{-1}(E) = \cup_{e \in E} f^{-1}(e)$.*

	α	$f(\alpha)$
1.	$A \sqsubseteq B_1 \sqcap \cdots B_n$	$(\{N_A\}, \{N_{B_i}\}), 1 \le i \le n$
2.	$A \equiv B_1 \sqcap \cdots B_m$	$(\{N_A\}, \{N_{B_i}\}), 1 \le i \le m$ $(\{N_{B_1}, \cdots, N_{B_m}\}, \{N_A\})$
3.	$A \sqsubseteq \exists r.B$	$(\{N_A\}, \{N_{\exists r.B}\})$
4.	$A \equiv \exists r.B$	$(\{N_{\exists r.B}\}, \{N_A\})$ and $(\{N_A\}, \{N_{\exists r.B}\})$
5.	$r \sqsubseteq s$	$(\{N_r\}, \{N_s\})$ and $(\{N_{\exists r.A}\}, \{N_{\exists s.A}\})$ for all A
6.	$r \circ s \sqsubseteq t$	$(\{N_r, N_s\}, \{N_s, N_t\})$

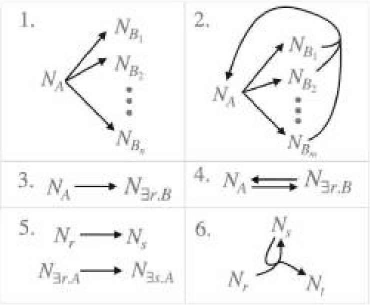


Fig. 1. Definition of f (left) and graphical illustrations of $f(\alpha)$ (right)

Notice that, the hyperedges associated with $A \equiv B_1 \sqcap \cdots \sqcap B_m$ are (i) the hyperedge $(\{N_{B_1}, \cdots, N_{B_m}\}, \{N_A\})$ and (2) of course, the edges corresponding to $A \sqsubseteq B_1 \sqcap \cdots \sqcap B_m$.

Example 4 (Example 1 cont'd). *The hypergraph $\mathcal{G}_\mathcal{O}$ for $\mathcal{O}$ is shown in Fig. 2, where $e_0 = (\{N_C\}, \{N_A\})$, $e_1 = (\{N_A\}, \{N_D\})$, $e_2 = (\{N_D\}, \{N_{\exists r.E}\})$, etc. Also, $f^{-1}(e_0) = C \sqsubseteq A$, $f^{-1}(e_1) = A \sqsubseteq D$, and $f^{-1}(e_2) = D \sqsubseteq \exists r.E$, etc.*

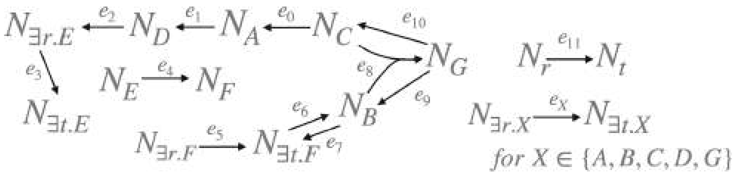


Fig. 2. The hypergraph associated with the ontology $\mathcal{O}$.

As for graphs, a path (next called **regular path**) from nodes N_1 to N_2 in a hypergraph is a sequence of edges:

$$e_0 = (S_1^0, S_2^0), e_1 = (S_1^1, S_2^1), \cdots, e_n = (S_1^n, S_2^n) \tag{1}$$

where $N_1 \in S_1^0, N_2 \in S_2^n$ and $S_2^{i-1} = S_1^i, 1 \le i \le n$. Next, the **existence** of a regular path from N_X to N_Y in a hypergraph $\mathcal{G}_\mathcal{O}$ is denoted $N_X \rightsquigarrow N_Y$. Now, we introduce hypergraph-based inferences which are based on the existence of regular paths as follows:

Table 2. H-rules over $\mathcal{G}_\mathcal{O} = (\mathcal{V}_\mathcal{O}, \mathcal{E}_\mathcal{O})$.

$$\mathcal{H}_0 : \frac{N_X \rightsquigarrow N_Y}{N_X \overset{h}{\rightsquigarrow} N_Y} \qquad \mathcal{H}_2 : \frac{N_X \overset{h}{\rightsquigarrow} N_{\exists r.B_1},\quad N_{B_1} \overset{h}{\rightsquigarrow} N_{B_2},\quad N_{\exists r.B_2} \rightsquigarrow N_Y}{N_X \overset{h}{\rightsquigarrow} N_Y}$$

$$\mathcal{H}_1 : \frac{N_X \overset{h}{\rightsquigarrow} N_{B_1}, \cdots, N_X \overset{h}{\rightsquigarrow} N_{B_m},\quad N_A \rightsquigarrow N_Y,\quad e}{N_X \overset{h}{\rightsquigarrow} N_Y} : e{=}(\{N_{B_1}, \cdots, N_{B_m}\}, \{N_A\}){\in}\mathcal{E}_\mathcal{O}$$

$$\mathcal{H}_3 : \frac{N_X \overset{h}{\rightsquigarrow} N_{\exists r.A_1},\quad N_{A_1} \overset{h}{\rightsquigarrow} N_{\exists s.A_2},\quad N_{\exists t.A_2} \rightsquigarrow N_Y,\quad e}{N_X \overset{h}{\rightsquigarrow} N_Y} : e{=}(\{N_r, N_s\}, \{N_s, N_t\}){\in}\mathcal{E}_\mathcal{O}$$

Definition 5. *Given a hypergraph $\mathcal{G}_\mathcal{O}$, Table 2 gives a set of inference rules called **H-rules**. Inferences based on H-rules are called **H-inferences**. Next, we denote by $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_Y$ (or simply $N_X \overset{h}{\rightsquigarrow} N_Y$) the fact that $N_X \overset{h}{\rightsquigarrow} N_Y$ can be derived from $\mathcal{G}_\mathcal{O}$ using the H-inferences.*

Example 6 (Example 4 cont'd). *As shown in Fig. 2, we have $N_A \rightsquigarrow N_{\exists r.E}$, $N_E \rightsquigarrow N_F$, $N_{\exists r.F} \rightsquigarrow N_B$ from the existence of regular paths. Then we can derive $N_A \overset{h}{\rightsquigarrow} N_B$ from $\mathcal{G}_\mathcal{O}$ by the H-rules $\mathcal{H}_0$, $\mathcal{H}_0$ and $\mathcal{H}_2$ which generate the H-inferences ρ^1, ρ^2, ρ^3, where $\rho^1 = \langle\{N_A \rightsquigarrow N_{\exists r.E}\}, N_A \overset{h}{\rightsquigarrow} N_{\exists r.E}\rangle$, $\rho^2 = \langle\{N_E \rightsquigarrow N_F\}, N_E \overset{h}{\rightsquigarrow} N_F\rangle$ and $\rho^3 = \langle\{N_A \overset{h}{\rightsquigarrow} N_{\exists r.E}, N_E \overset{h}{\rightsquigarrow} N_F, N_{\exists r.F} \rightsquigarrow N_B\}, N_A \overset{h}{\rightsquigarrow} N_B\rangle$, respectively.*

Note that the first rule $\mathcal{H}_0$, the initialization rule, makes regular paths the elementary components of H-rules. Moreover, Proposition 7 formally states that, in our H-inference system, we do not need to add the transitive inference rule:

$$\frac{N_X \overset{h}{\rightsquigarrow} N_Z, N_Z \overset{h}{\rightsquigarrow} N_Y}{N_X \overset{h}{\rightsquigarrow} N_Y}.$$

Proposition 7. *If $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_Z$ and $\mathcal{O} \vdash_h N_Z \overset{h}{\rightsquigarrow} N_Y$ then $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_Y$.*

3.2 Completeness and Soundness of H-Inferences

The following result is the main result of this section. It states the equivalence of $N_X \overset{h}{\rightsquigarrow} N_Y$ derivation (by Table 2) and ontology entailment for $X \sqsubseteq Y$, and thus states that our H-rules are sound and complete for $\mathcal{EL}^+$-ontology.

Theorem 8. *If $\mathcal{O}$ is an $\mathcal{EL}^+$-ontology, then $\mathcal{O} \models X \sqsubseteq Y$ iff $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_Y$, where X, Y are concepts of either form A or $\exists r.B$.*

Proof. "$\Leftarrow$" is obvious by induction over Table 2 and the fact that $N_X \rightsquigarrow N_Y$ implies $\mathcal{O} \models X \sqsubseteq Y$, so we only need to prove the direction "$\Rightarrow$".

Assume that $\mathcal{O} \models X \sqsubseteq Y$. We consider two cases:

Case 1. We assume $\mathcal{O} \vdash X \sqsubseteq Y$[1]. Let $d(X, Y)$ be the length of one shortest derivation of $X \sqsubseteq Y$ from $\mathcal{O}$ using Table 1. We prove "⇒" by induction on $d(X, Y)$.

- **Assume** $d(X, Y) = 0$**.** In this case $\mathcal{O}$ must contain axioms of the form $X \equiv Y \sqcap \cdots$ or $X \sqsubseteq Y \sqcap \cdots$. Clearly we have $N_X \rightsquigarrow N_Y$ thus $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_Y$.
- **Assuming "⇒" holds when** $d(X, Y) < k$**, let us prove "⇒" holds for** $d(X, Y) = k$**.** Suppose ρ^{last} is the last inference in one shortest derivation of $X \sqsubseteq Y$ using Table 1. Two cases arise:
  1. Assume ρ^{last} is generated by $\mathcal{R}_1(n > 1), \mathcal{R}_3$ or $\mathcal{R}_4(n = 2)$. For example, assume $\rho^{last} = \langle \{X \sqsubseteq \exists r.B_1, B_1 \sqsubseteq B_2, \exists r.B_2 \sqsubseteq Y\}, X \sqsubseteq Y \rangle$ comes from $\mathcal{R}_3$. We have $d(X, \exists r.B_1), d(B_1, B_2), d(\exists r.B_2, Y) < k$ because their corresponding subsumptions can be derived without ρ^{last}. By the assumption $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_{\exists r.B_1}, N_{B_1} \overset{h}{\rightsquigarrow} N_{B_2}, N_{\exists r.B_2} \overset{h}{\rightsquigarrow} N_Y$. Then we have $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_{\exists r.B_2}$ by first deriving $N_X \overset{h}{\rightsquigarrow} N_{\exists r.B_1}, N_{B_1} \overset{h}{\rightsquigarrow} N_{B_2}$, and then applying H-inference:
     $$\rho^{new} = \langle \{N_X \overset{h}{\rightsquigarrow} N_{\exists r.B_1}, N_{B_1} \overset{h}{\rightsquigarrow} N_{B_2}, N_{\exists r.B_2} \rightsquigarrow N_{\exists r.B_2}\}, N_X \overset{h}{\rightsquigarrow} N_{\exists r.B_2} \rangle.$$
     Then $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_Y$ by Proposition 7 since $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_{\exists r.B_2}, N_{\exists r.B_2} \overset{h}{\rightsquigarrow} N_B$. The argument also holds for $\mathcal{R}_1(n > 1)$(or $\mathcal{R}_4(n = 2)$) by applying $\mathcal{H}_1$ (or $\mathcal{H}_3$) instead of $\mathcal{H}_2$.
  2. Assume ρ^{last} is generated by $\mathcal{R}_1(n = 1), \mathcal{R}_2$ or $\mathcal{R}_4(n = 1)$. Then, in each case, we have ρ^{last} has the form $\langle \{X \sqsubseteq Z, Z \sqsubseteq Y\}, X \sqsubseteq Y \rangle$. As in case 1, we have $d(X, Z), d(Z, Y) < k$. By the assumption, $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_Z, N_Z \overset{h}{\rightsquigarrow} N_Y$, then $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_Y$ by Proposition 7.

Case 2. If $\mathcal{O} \vdash X \sqsubseteq Y$ does not hold, then X or Y is not atomic. In this case, we introduce new axioms $A \equiv X$, $B \equiv Y$ with new atomic concepts A, B and denote the extended ontology by $\mathcal{O}'$. Clearly, $\mathcal{O}' \models A \sqsubseteq B$ and thus $\mathcal{O}' \vdash A \sqsubseteq B$ since Table 1 is sound and complete. Therefore, we have $\mathcal{O}' \vdash_h N_A \overset{h}{\rightsquigarrow} N_B$ by the same arguments as above. Now, notice that $\mathcal{G}_{\mathcal{O}'}$ is obtained from $\mathcal{G}_{\mathcal{O}}$ by adding 4 edges: $(\{N_A\}, \{N_X\}), (\{N_X\}, \{N_A\}), (\{N_B\}, \{N_Y\})$ and $(\{N_Y\}, \{N_B\})$, thus we have $\mathcal{O}' \vdash_h N_A \overset{h}{\rightsquigarrow} N_B$ iff $\mathcal{O} \vdash_h N_X \overset{h}{\rightsquigarrow} N_Y$.

3.3 Extracting Justifications from $\mathcal{G}_{\mathcal{O}}$

Now, we formally define H-paths as a hypergraph representation of classical derivations based on H-rules. The reader should pay attention to the fact that H-paths are not classical hyperpaths [7]. Next, for the sake of homogeneity, we consider a regular path from N_X to N_Y as the set of its edges and denote it as $P_{X,Y}$.

[1] The reader should recall that the equivalence ($\mathcal{O} \models X \sqsubseteq Y$ iff $\mathcal{O} \vdash X \sqsubseteq Y$) only holds when X and Y are atomic concepts wrt. the inference system presented in Table 1.

Definition 9 (H-paths). *In the hypergraph $\mathcal{G}_{\mathcal{O}}$, an H-path $H_{X,Y}$ from N_X to N_Y is a set of edges recursively generated by the following composition rules:*

0. A regular path $P_{X,Y}$ is an H-path from N_X to N_Y;
1. If $e = (\{N_{B_1}, \cdots, N_{B_m}\}, \{N_A\}) \in \mathcal{V}_{\mathcal{O}}$, if H_{X,B_i} are H-paths for $i = 1..m$, and if $P_{A,Y}$ is a regular path, then $H_{X,B_1} \cup \cdots \cup H_{X,B_m} \cup P_{A,Y} \cup \{e\}$ is an H-path from N_X to N_Y;
2. If $H_{X,\exists r.B_1}, H_{B_1,B_2}$ are H-paths and $P_{\exists r.B_2,Y}$ is a regular path, then $H_{X,\exists r.B_1} \cup H_{B_1,B_2} \cup P_{\exists r.B_2,Y}$ is an H-path from N_X to N_Y;
3. If $e = (\{N_r, N_s\}, \{N_s, N_t\}) \in \mathcal{V}_{\mathcal{O}}$, if $H_{X,\exists r.A_1}, H_{A_1,\exists s.A_2}$ are H-paths and if $P_{\exists t.A_2,B}$ is a regular path, then $H_{X,\exists r.A_1} \cup H_{A_1,\exists s.A_2} \cup P_{\exists t.A_2,B} \cup \{e\}$ is an H-path from N_X to N_Y.

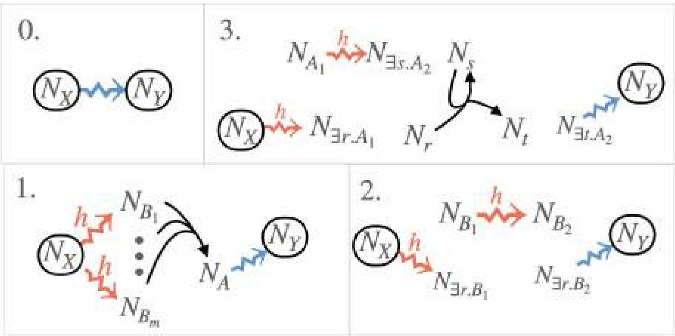


Fig. 3. Structure of H-paths from N_X to N_Y

Figure 3 gives an illustration of H-paths: the blue arrows $\rightsquigarrow$ correspond to regular paths, and the red ones $\overset{h}{\rightsquigarrow}$ to H-paths. It is straightforward to compare composition rules building H-paths with H-rules building derivations in Table 2. One may also consider H-paths as deviation-trees with leaves corresponding to the edges in $\mathcal{G}_{\mathcal{O}}$. However, our approach provides a more direct characterization of justifications as shown in Theorem 10.

We say that an H-path $H_{X,Y}$ is **minimal** if there is no H-path $H'_{X,Y}$ such that $H'_{X,Y} \subset H_{X,Y}$.

Now, we are ready to explain how H-paths and justifications are related. We can compute justifications from minimal H-paths as stated below:

Theorem 10. *Given X, Y of either form A or $\exists r.B$. Let*

$$\mathcal{S} = \{f^{-1}(H_{X,Y}) \mid H_{X,Y} \text{ is a minimal H-path from } N_X \text{ to } N_Y\}.$$

Then $\mathcal{J}_{\mathcal{O}}(X \sqsubseteq Y) = \{s \in \mathcal{S} \mid s' \not\subset s, \forall s' \in \mathcal{S}\}$. That is, all justifications for $X \sqsubseteq Y$ are the minimal subsets in $\mathcal{S}$.

Proof. For any justification $\mathcal{O}'$ of $X \sqsubseteq Y$, there exists a minimal H-path $H_{X,Y}$ such that $\mathcal{O}' = f^{-1}(H_{X,Y})$. The reason is that, since $\mathcal{O}' \models X \sqsubseteq Y$, there exists an H-path $H_{X,Y}$ from N_X to N_Y on $\mathcal{G}_{\mathcal{O}'}$ by Theorem 8. Without loss of generality, we can assume $H_{X,Y}$ is minimal on $\mathcal{G}_{\mathcal{O}'}$, then it is also minimal on $\mathcal{G}_{\mathcal{O}}$ since $\mathcal{G}_{\mathcal{O}'}$ is a sub-graph of $\mathcal{G}_{\mathcal{O}}$. We have $\mathcal{O}' = f^{-1}(H_{X,Y})$ because otherwise there exists $\mathcal{O}'' \subsetneq \mathcal{O}'$ such that $\mathcal{O}'' = f^{-1}(H_{X,Y})$, and thus $\mathcal{O}'' \models X \sqsubseteq Y$ by Theorem 8 again. Therefore, $\mathcal{O}'$ is not a justification. Contradiction.

Now, we know $\mathcal{S}$ contains all justifications for $X \sqsubseteq Y$. Moreover, $f^{-1}(H_{X,Y}) \models X \sqsubseteq Y$ for any H-path $H_{X,Y}$. Therefore, we have $\mathcal{J}_{\mathcal{O}}(X \sqsubseteq Y) = \{s \in \mathcal{S} \mid s' \not\subset s, \forall s' \in \mathcal{S}\}$ by the definition of justifications.

Example 11. (Example 4 cont'd). *The regular paths from N_A to $N_{\exists r.E}$ and from N_E to N_F produce two H-paths $H_{A,\exists rE} = \{e_1, e_2, e_3\}$ and $H_{E,F} = \{e_4\}$. Then, applying the third composition rule with $H_{A,\exists rE}, H_{E,F}$ and $P_{\exists r.F,B} = \{e_6\}$, we get $H_{A,B} = \{e_1, e_2, e_3, e_4, e_6\}$, which is the unique H-path from N_A to N_B. Thus, by Theorem 10, we have $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$, the only justification for $A \sqsubseteq B$.*

4 Implementation: Computing Justifications

4.1 SAT-Based Method

In this section, we describe briefly how PULi [14], the state-of-the-art *glass-box* algorithm, proceeds. Given an ontology $\mathcal{O}$, computing $J_{\mathcal{O}}(X \sqsubseteq Y)$ is done through 2 steps: (1) tracing *a complete set* for $X \sqsubseteq Y$, (2) using resolution to extract the justifications from *the complete set*. The following example illustrates both steps:

Example 12 (Example 1 cont'd). *Let us compute $J_{\mathcal{O}}(G \sqsubseteq D)$ using PULi's method.*

1. *Using the goal-directed tracing algorithm in [12], the first step produces a complete set of inferences*[2] *$\{\rho_1, \rho_2\}$ for $G \sqsubseteq D$, where $\rho_1 = \langle\{G \sqsubseteq C, C \sqsubseteq A\}, G \sqsubseteq A\rangle, \rho_2 = \langle\{G \sqsubseteq A, A \sqsubseteq D\}, G \sqsubseteq D\rangle$.*
2. *This step is again composed of two parts:*
   (a) *The first part proceeds to the translation of the inferences into clauses. Let us denote $\overline{p}_1$:$G \sqsubseteq C$, $\overline{p}_2$:$C \sqsubseteq A$, $\overline{p}_3$:$A \sqsubseteq D$, p_4:$G \sqsubseteq A$, p_5:$G \sqsubseteq D$. Here the literals $\overline{p}_1, \overline{p}_2, \overline{p}_3$ (with a bar) are called* ***answer literals*** *as they correspond to the axioms a_6, a_7, a_1 in $\mathcal{O}$. Thus, we obtain $\mathcal{C} = \{\neg\overline{p}_1 \vee \neg\overline{p}_2 \vee p_4, \neg p_4 \vee \neg\overline{p}_3 \vee p_5\}$ by rewriting the inferences ρ_1, ρ_2 as clauses.*
   (b) *Secondly, a new clause $\neg p_5$ is added to $\mathcal{C}$, where p_5 corresponds to the conclusion $G \sqsubseteq D$, and resolution is applied over $\mathcal{C}$. The set of all justifications $J_{\mathcal{O}}(G \sqsubseteq D)$ is obtained by considering (i) the clauses formed of*

[2] For the sake of simplicity, we use the inference rules in Table 1 although PULi uses a slightly different set of inference rules [13].

Algorithm 1: `minH`

```
  input  : $X \sqsubseteq Y$
  output: J: $\mathcal{J}_{\mathcal{O}}(X \sqsubseteq Y)$.
1 J ← $\emptyset$;
2 $\mathcal{U}$ ← CompleteH($N_X \overset{h}{\rightsquigarrow} N_Y$);
3 min_hpaths ← resolution(clauses($\mathcal{U}$));
4 for h ∈ min_hpaths do
5     if $f^{-1}$(h') $\not\subset$ $f^{-1}$(h) for any h' ∈ min_hpaths then
6         J.add($f^{-1}$(h))
7     end
8 end
```

answer literals only and (ii) among them keeping the minimal ones[3]*. In this example, after the resolution phase, the only clause that consists of merely answer literals is* $\neg\overline{p}_1 \vee \neg\overline{p}_2 \vee \neg\overline{p}_3$*. Thus, the set of all justifications is* $J_{\mathcal{O}}(G \sqsubseteq D) = \{\{a_1, a_6, a_7\}\}$*.*

Our method for computing justifications follows the same steps as PULi although here the major difference is that the first step computes a complete set of H-inferences instead of a complete set of inferences wrt. Table 1.

4.2 Computing Justification by Minimal H-Paths

In this section, given an ontology $\mathcal{O}$ and its associated hypergraph $\mathcal{G}_{\mathcal{O}}$, we present `minH` (Algorithm 1) that computes all justifications for $X_0 \sqsubseteq Y_0$ using the minimal H-paths from N_{X_0} to N_{Y_0} over $\mathcal{G}_{\mathcal{O}}$. The algorithm `minH` proceeds in two steps described below.

Step 1. First, at Line 2, `minH` computes a *complete set* of inferences $\mathcal{U}$ for $N_{X_0} \overset{h}{\rightsquigarrow} N_{Y_0}$ using `CompleteH` (See Algorithm 2). Here, $\mathcal{U}$ is complete in the sense that for any H-path $H_{X,Y}$, we can derive $N_X \overset{h}{\rightsquigarrow} N_Y$ using inferences in $\mathcal{U}$ from the edge set $H_{X,Y}$. `CompleteH` computes $\mathcal{U}$ as follows:

- **Line 3–12 of Algorithm 2:** The recursive application of `trace_one_turn` (See Algorithm 3) outputs the set of all H-inferences whose conclusion is the given input $N_{X_1} \overset{h}{\rightsquigarrow} N_{Y_1}$;
- **Line 13–17 of Algorithm 2:** Let `path` be the depth-first search algorithm that computes all regular paths from N_X to N_Y in $\mathcal{G}_{\mathcal{O}}$ with input (N_X, N_Y). Intuitively, the purpose is to shift inferences from regular paths to edges.

Step 2. Then Algorithm `minH` computes all justifications for $X_0 \sqsubseteq Y_0$ as follows:

[3] Here a clause c is smaller than c_1 if all the literals of c are in c_1.

Algorithm 2: `CompleteH`

```
   input  : $N_X \overset{h}{\rightsquigarrow} N_Y$
   output: $\mathcal{U}$: a complete set of inferences for $N_X \overset{h}{\rightsquigarrow} N_Y$.
1  $\mathcal{U}$, history, Q $\leftarrow \emptyset$ ;                               // Q is a queue
2  Q.$add(N_X \overset{h}{\rightsquigarrow} N_Y)$;
3  while Q $\neq \emptyset$ do
4      $N_{X_1} \overset{h}{\rightsquigarrow} N_{Y_1} \leftarrow$ Q.$takeNext()$;
5      history.$add(N_{X_1} \overset{h}{\rightsquigarrow} N_{Y_1})$;
6      $\mathcal{U} \leftarrow \mathcal{U} \bigcup$ trace_one_turn($N_{X_1} \overset{h}{\rightsquigarrow} N_{Y_1}$);
7      for $N_{X_2} \overset{h}{\rightsquigarrow} N_{Y_2}$ appearing in trace_one_turn($N_{X_1} \overset{h}{\rightsquigarrow} N_{Y_1}$) do
8          if $N_{X_2} \overset{h}{\rightsquigarrow} N_{Y_2} \notin$ history and $N_{X_2} \overset{h}{\rightsquigarrow} N_{Y_2} \notin$ Q then
9              Q.add($N_{X_2} \overset{h}{\rightsquigarrow} N_{Y_2}$)
10         end
11     end
12 end
13 for $N_{X_2} \rightsquigarrow N_{Y_2}$ appearing in $\mathcal{U}$ do
14     for $p$={$e_1, e_2, \cdots, e_n$} $\in$path($N_{X_2}, N_{Y_2}$) do
15         $\mathcal{U}.add(\langle\{e_1, e_2, \cdots, e_n\}, N_{X_2} \rightsquigarrow N_{Y_2}\rangle)$;
16     end
17 end
```

- **Line 3 of Algorithm 1:** It computes all minimal H-paths from N_{X_0} to N_{Y_0} using resolution, which is developed by PULi[4], over the clauses generated from $\mathcal{U}$ as illustrated in Sect. 4.1. Here, a literal p is associated with each edge e, each $N_X \overset{h}{\rightsquigarrow} N_Y$, and each $N_X \rightsquigarrow N_Y$ in $\mathcal{U}$. The answer literals are those associated with edges.
- **Line 4–8 of Algorithm 1:** It computes justifications by mapping back all the minimal H-paths and select the subset-minimal sets as stated in Theorem 10.

Example 13 (Example 4 cont'd). *Assume $X_0 = G$ and $Y_0 = D$ are the input of* `minH`*. Then at line 2 of* `minH`*, we have $\mathcal{U} = \{\rho^1, \rho^2\}$, where $\rho^1 = \langle\{N_G \rightsquigarrow N_D\}, N_G \overset{h}{\rightsquigarrow} N_D\rangle$ is H-inference obtained by* `CompleteH` *(line 3–12) and $\rho^2 = \langle\{e_0, e_1, e_8\}, N_G \rightsquigarrow N_D\rangle$ is produced from regular paths obtained by* `CompleteH` *(line 13–17). Let us denote $\overline{p}_0$:e_0, $\overline{p}_1$:e_1, $\overline{p}_2$:e_8 as answer literals and p_3:$N_G \rightsquigarrow N_D$, p_4:$N_G \overset{h}{\rightsquigarrow} N_D$. Then* `clauses`*($\mathcal{U}$) $= \{\neg p_3 \vee p_4,\ \neg\overline{p}_0 \vee \neg\overline{p}_1 \vee \neg\overline{p}_2 \vee p_3\}$.*

By resolution over `clauses`*($\mathcal{U}$), we obtain* `min_hpaths` *$= \{\{e_0, e_1, e_8\}\}$ at line 3 of* `minH`*. Then the output of* `minH` *is $J = \{\{a_1, a_6, a_7\}\}$, which is the set of all justifications for $G \sqsubseteq D$.*

[4] Available at https://github.com/liveontologies/pinpointing-experiments.

Algorithm 3: `trace_one_turn`

```
input  : $N_X \overset{h}{\rightsquigarrow} N_Y$
output: the set result of all H-inferences whose conclusion is $N_X \overset{h}{\rightsquigarrow} N_Y$.
1  result ← $\emptyset$;
2  $\mathcal{P}_1(X,Y) \leftarrow \{(\{N_{B_1},\cdots,N_{B_m}\},\{N_A\}) \in \mathcal{E}^{\mathcal{O}} \mid \mathcal{O} \models X \sqsubseteq A \sqsubseteq Y\}$;
3  for $(\{N_{B_1},\cdots,N_{B_m}\},\{N_A\}) \in \mathcal{P}_1(X,Y)$ do
4      if path($N_A, N_Y$)$\neq \emptyset$ or $Y{=}A$ then
5          result.add($\langle\{N_X \overset{h}{\rightsquigarrow} N_{B_1},\cdots,N_X \overset{h}{\rightsquigarrow} N_{B_m}, N_A \rightsquigarrow N_Y\}, N_X \overset{h}{\rightsquigarrow} N_Y\rangle$) ;
6      end
7  end
8  $\mathcal{P}_2(X,Y) \leftarrow \{(r,B_1,B_2) \mid \mathcal{O} \models X \sqsubseteq \exists r.B_1,\ B_1 \sqsubseteq B_2, \exists r.B_2 \sqsubseteq Y\}$;
9  for $(r,B_1,B_2) \in \mathcal{P}_2(X,Y)$ do
10     if path($N_{\exists r.B_2}, N_Y$)$\neq \emptyset$ or $Y{=}\exists r.B_2$ then
11         result.add($\langle\{N_X \overset{h}{\rightsquigarrow} N_{\exists r.B_1}, N_{B_1} \overset{h}{\rightsquigarrow} N_{B_2}, N_{\exists r.B_2} \rightsquigarrow N_Y\}, N_X \overset{h}{\rightsquigarrow} N_Y\rangle$);
12     end
13 end
14 $\mathcal{P}_3(X,Y) \leftarrow \{(r,s,t,A_1,A_2) \mid r \circ s \sqsubseteq t \in \mathcal{O},\ \mathcal{O} \models X \sqsubseteq \exists r.A_1, A_1 \sqsubseteq \exists s.A_2, \exists t.A_2 \sqsubseteq Y\}$;
   for $(r,s,t,A_1,A_2) \in \mathcal{P}_3(X,Y)$ do
15     if path($N_{\exists t.A_2}, N_Y$)$\neq \emptyset$ or $Y{=}\exists t.A_2$ then
16         result.add($\langle\{N_X \overset{h}{\rightsquigarrow} N_{\exists r.A_1}, N_{A_1} \overset{h}{\rightsquigarrow} N_{\exists s.A_2}, N_{\exists t.A_2} \rightsquigarrow N_Y, (\{N_r,N_t\},\{N_s,N_t\})\},$
             $\{N_s,N_t\})\}, N_X \overset{h}{\rightsquigarrow} N_Y\rangle$);
17     end
18 end
```

4.3 Optimization

Below we present two optimizations that have been implemented in order to accelerate the computation of all justifications.

1. In Algorithm 3, for the H-inference added at Line 5, we require that there exists at least one regular path from N_A to N_Y that does not contain an edge $e_i = (\{N_A\},\{N_{B_i}\})$ for some $1 \leq i \leq m$. Otherwise, as shown in Fig. 4, H-paths corresponding to this H-inference are not minimal, as they all contain one H-path from N_X to N_Y of the form $H_{X,B_i} \cup (P_{A,Y} - \{e_i\})$. In the same spirit, we require that the H-path from N_X to N_{B_i} does not pass by N_A.
2. If we have an H-path $H_{A,B} = H_{A,\exists r.B_1} \cup H_{B_1,B_2} \cup P_{\exists r.B_2,B}$ where
$$H_{A,\exists r.B_1} = H_{A,\exists r.C} \cup H_{C,B_1}. \tag{2}$$
then $H_{C,B_2} = H_{C,B_1} \cup H_{B_1,B_2}$ is also an H-path and $H_{A,B} = H_{A,\exists r.C} \cup H_{C,B_2} \cup P_{\exists r.B_2,B}$. The two different ways to decompose $H_{A,B}$ above are already considered in Line 8 when executing Algorithm 3 with the input $N_A \overset{h}{\rightsquigarrow} N_B$. It means that the decomposition (2) is redundant. We can avoid such redundancy by requiring $\exists r.B_2 \neq Y$ at Line 11.

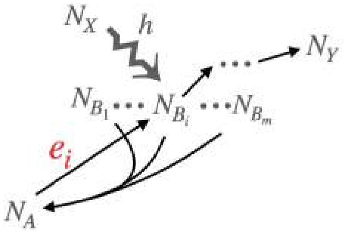


Fig. 4. Illustration of Optimization 1

5 Experiments

To evaluate and validate our approach, we compare `minH`[5] with PULi [14], the state-of-the-art algorithm for computing justifications at this moment. Both methods compute all justifications based on resolution but with different inference rules generated in different ways. PULi uses a complete set (next denoted by *elk*) generated by the ELK reasoner [13], which uses inference rules slightly different from those in Table 1. Our method uses the complete set $\mathcal{U}$ generated by Step 1 of `minH`, described in Sect. 4.2. To analyze the performance of our setting, we make the following two measures: (1) we compare the size of *elk* with that of $\mathcal{U}$, (2) we compare the time cost of PULi with that of `minH`. All the experiments were conducted on a machine with an INTEL Xeon 2.6 GHz and 128 GiB of RAM.

The experiments were processed with four different ontologies[6]: go-plus, galen7, SnomedCT (version Jan. 2015 and Jan. 2021). All the non-$\mathcal{EL}^+$ axioms are deleted. Here, go-plus, galen7 are the same ontologies used in [14]. We denote the four ontologies above by go-plus, galen7, snt2015 and snt2021. The number of axioms, concepts, relations, and queries for each ontology are shown in Table 3.

Next a **query** refers to a *direct subsumption*[7] $A \sqsubseteq B$. In our experiments, for the four ontologies, the set of all justifications $J_{\mathcal{O}}(A \sqsubseteq B)$ is computed for each query $A \sqsubseteq B$. A query $A \sqsubseteq B$ is called **trivial** iff all minimal H-paths from N_A to N_B are regular paths, otherwise, the query is **non-trivial**.

Comparing Complete Sets: $\mathcal{U}$ vs. *elk*. We summarize our results in Table 4 and Fig. 5. Table 4 shows that on all four ontologies, $\mathcal{U}$ is much smaller than *elk* on average. Especially on galen7, the difference between *elk* and $\mathcal{U}$ is even up to 50 times. The gap is even more significant for the median value since a large part of the queries is trivial. However, the gap is much smaller for the maximal number. On snt2021, the largest $\mathcal{U}$ in size is three times larger than that of *elk*.

[5] A prototype is available at https://gitlab.lisn.upsaclay.fr/yang/minH.

[6] Available at https://osf.io/9sj8n/, https://www.snomed.org/.

[7] i.e., $\mathcal{O} \models A \sqsubseteq B$ and there is no other atomic concept A' such that $\mathcal{O} \models A \sqsubseteq A', A' \sqsubseteq B$. Direct subsumptions can be computed by a reasoner supporting ontology classification.

Table 3. Summary of sizes of the input ontologies.

	go-plus	galen7	snt2015	snt2021
#axioms	105557	44475	311466	362638
#concepts	57173	28482	311480	361226
#roles	157	964	58	132
#queries	90443	91332	461854	566797

Table 4. Summary of size of *elk*, $\mathcal{U}$.

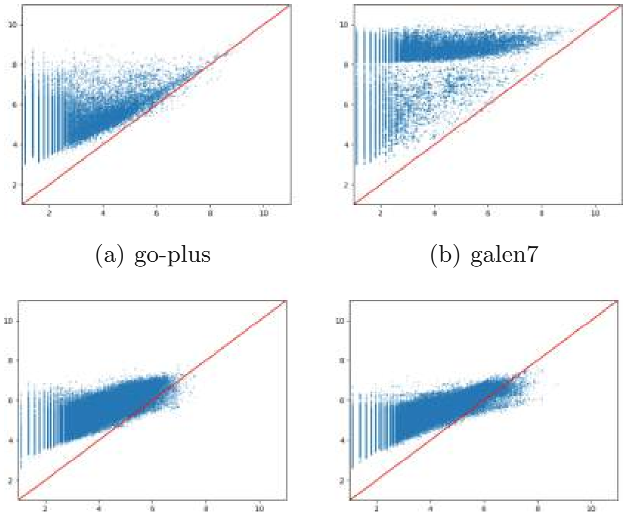
		go-plus	galen7	snt2015	snt2021
elk	average	166.9	3602.0	114.7	67.3
	median	43.0	3648.0	10.0	31.0
	max	7919.0	81501.0	2357	2226
$\mathcal{U}$	average	34.2	74.6	29.4	19.4
	median	4.0	5.0	1.0	3.0
	max	7772	24103	2002	6452
#non-trivial query		50272	62470	195082	304321

In Fig. 5, for a given query, if the complete set *elk* contains fewer inference rules than $\mathcal{U}$, the corresponding blue point is below the red line. The percentage of such cases are: 0.34% for go-plus, 0.066% for galen7, 0.79% for snt2015, and 1.01% for snt2021. It means that for most of the queries, the corresponding $\mathcal{U}$ is smaller than *elk*.

As shown in Table 4 and in Fig. 5, sometimes `minH` generates bigger complete set $\mathcal{U}$ than PULi. It may happen when, for example, there might be exponentially many different regular paths occurring in the computation process of `minH`. Therefore, `minH` could produce a huge complete set. Also, $\mathcal{U}$ can be bigger than *elk* when all the regular paths involved are simple. For example, if all regular paths contain only one edge, then the complete set $\mathcal{U}$ includes many clauses of the form $\neg p_e \vee p_{N_A \rightsquigarrow N_B}$, which happens because H-rules use regular paths. Indeed, the clause $\neg p_e \vee p_{N_A \rightsquigarrow N_B}$ is redundant since we can omit this clause by replacing $p_{N_A \rightsquigarrow N_B}$ by p_e. For *elk*, this does not happen.

Comparing Time Cost: `minH` vs. PULi. In the following, we only compare the time cost on non-trivial queries. For trivial queries, all H-path are regular paths. Thus all the justifications have already been enumerated by `path` in `minH`. It is also easy to compute all the justifications for trivial queries for PULi.

We set a limit of 60 s for each query. The timed-out queries contribute of 60 s to the total time cost. To compare `minH` with PULi, we test all three different strategies, *threshold, top down, bottom up* of the resolution algorithm proposed in [14]. We summarize in Table 5 the total time cost (top) and the timed-out queries (bottom). Figure 6 gives the comparisons over queries that are successful for both `minH` and PULi.



(a) go-plus

(b) galen7

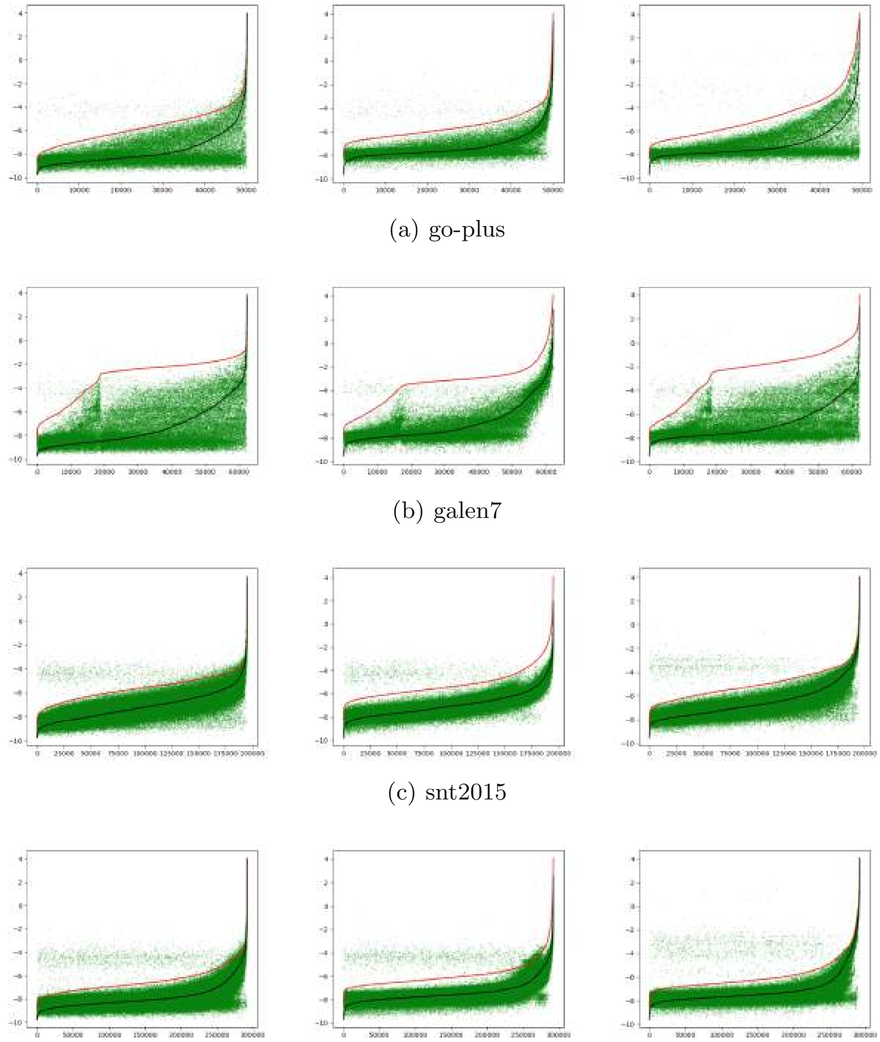
(c) snt2015

(d) snt2021

Fig. 5. Each blue point has coordinate $(\log(\#|\mathcal{U}|), \log(\#|elk|))$, where $\mathcal{U}$, *elk* are generated from a non-trivial query, the red line is $x = y$. (Color figure online)

As shown in Table 5, when using the threshold strategy, `minH` is more time consuming in total (+5%) on snt2021, and `minH` has more timed-out queries than PULi on snt2015 and snt2021. This is in part due to the fact that $\mathcal{U}$ is larger than *elk* for relatively many queries on snt2015 and snt2021 as shown in Fig. 5. For the remaining 11 cases, `minH` performs better than PULi in terms of total time cost and the number of timed-out queries. Especially on galen7, the gap between the two methods is even up to ten times for the total time cost. We can see from Table 5 that the threshold strategy performs the best for PULi on all four ontologies. This strategy is also the best strategy for `minH` except for galen7, for which the *bottom up* strategy is the best with `minH`.

For each strategy detailed in Fig. 6, the black curve (the ordered time costs of `minH` on successful queries) is always below the red curve (the ordered time costs of PULi on successful queries) for all the ontologies. This suggests that `minH` spends less time over successful queries. Also, most of the green points are below the red lines, which suggests that `minH` performs better than PULi most of the time for a given query. In some cases, we can see that PULi is more efficient than `minH`. One of the reasons might be as follows. Note that when computing justifications by resolution, we have to compare two different clauses and delete the redundant one (i.e., the non-minimal one). When regular paths are big, `minH` might be time consuming because of these comparisons.



(a) go-plus

(b) galen7

(c) snt2015

(d) snt2021

Fig. 6. For each line, the left, middle and right charts correspond to *threshold, top down, bottom up* strategies respectively. The y-axis is the log value of time(s). The red (resp. black) curve presents the ascending ordered (log value of) time cost of PULi (resp. `minH`). For a green point (x, y), e^y is the time cost of `minH` for the query corresponding to the red line point (x, y'). (Color figure online)

Table 5. Total time cost and number of timed-out queries.

		threshold	*top down*	*bottom up*
total times(s) (PULi/`minH`)	go-plus	8482.7/**7350.3**	16352.3/**8935.6**	73629.1/**17950.9**
	galen7	10796.2/**3681.4**	43372.9/**10607.9**	36300.9/**3156.3**
	snt2015	1956.8/**973.5**	13650.7/**1107.6**	15058.3/**11392.2**
	snt2021	**2116.1**/2222.6	11573.9/**2361.6**	19402.1/**17154.9**
timed-out queries (PULi/`minH`/both)	go-plus	116/**103** /93	202/**117**/114	935/**223**/223
	galen7	48/**43**/43	370/**123**/120	228/**38**/38
	snt2015	**0**/3/0	49/**3**/3	96/**88**/83
	snt2021	**2**/8/1	39/**9**/9	144/**133**/128

6 Conclusion

In this paper, we introduce and investigate a new set of sound and complete inference rules based on a hypergraph representation of ontologies. We design the algorithm `minH` that leverages these inference rules to compute all justifications for a given conclusion. The key of the performance of our method is that regular paths are used as elementary components of H-paths and this leads to reducing the size of complete sets because (1) rules are more compact than standard ones, (2) redundant inferences are captured and eliminated by regular paths (see Sect. 4.3). The efficiency of the algorithm `minH` has been validated by our experiments showing that it outperforms PULi in most of the cases.

There are still many possible extensions and applications of the hypergraph approach. For instance, to get even more compact inference rules, we could extend the notion of regular path to a more general one that will encapsulate the inference rule $\mathcal{H}_2$ in the same way as regular paths are encapsulated in H-rules. Moreover, we will try to apply our approach for other tasks like classification and to compute logical differences [15].

References

1. Arif, M.F., Mencía, C., Ignatiev, A., Manthey, N., Peñaloza, R., Marques-Silva, J.: BEACON: an efficient SAT-based tool for debugging $\mathcal{EL}^+$ ontologies. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 521–530. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_32
2. Arif, M.F., Mencía, C., Marques-Silva, J.: Efficient axiom pinpointing with EL2MCS. In: Hölldobler, S., Krötzsch, M., Peñaloza, R., Rudolph, S. (eds.) KI 2015. LNCS (LNAI), vol. 9324, pp. 225–233. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24489-1_17
3. Arif, M.F., Mencía, C., Marques-Silva, J.: Efficient MUS enumeration of horn formulae with applications to axiom pinpointing. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 324–342. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_24
4. Ausiello, G., Laura, L.: Directed hypergraphs: introduction and fundamental algorithms-a survey. Theor. Comput. Sci. **658**, 293–306 (2017)

5. Baader, F., Brandt, S., Lutz, C.: Pushing the EL envelope. In: IJCAI, vol. 5, pp. 364–369 (2005)
6. Chen, J., Ludwig, M., Ma, Y., Walther, D.: Zooming in on ontologies: minimal modules and best excerpts. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10587, pp. 173–189. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68288-4_11
7. Gallo, G., Longo, G., Pallottino, S., Nguyen, S.: Directed hypergraphs and applications. Discret. Appl. Math. **42**(2–3), 177–201 (1993)
8. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: theory and practice. J. Artif. Intell. Res. **31**, 273–318 (2008)
9. Ignatiev, A., Marques-Silva, J., Mencía, C., Peñaloza, R.: Debugging EL+ ontologies through horn MUS enumeration. In: Artale, A., Glimm, B., Kontchakov, R. (eds.) Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, 18–21 July 2017. CEUR Workshop Proceedings, vol. 1879. CEUR-WS.org (2017). http://ceur-ws.org/Vol-1879/paper54.pdf
10. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Aberer, K., et al. (eds.) ASWC/ISWC -2007. LNCS, vol. 4825, pp. 267–280. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_20
11. Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.: Debugging unsatisfiable classes in OWL ontologies. J. Web Semant. **3**(4), 268–293 (2005)
12. Kazakov, Y., Klinov, P.: Goal-directed tracing of inferences in EL ontologies. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8797, pp. 196–211. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11915-1_13
13. Kazakov, Y., Krötzsch, M., Simancik, F.: ELK reasoner: architecture and evaluation. In: ORE (2012)
14. Kazakov, Y., Skočovský, P.: Enumerating justifications using resolution. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS (LNAI), vol. 10900, pp. 609–626. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94205-6_40
15. Ludwig, M., Walther, D.: The logical difference for $\mathcal{ELH}^r$-terminologies using hypergraphs. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, 18–22 August 2014 - Including Prestigious Applications of Intelligent Systems (PAIS 2014). Frontiers in Artificial Intelligence and Applications, vol. 263, pp. 555–560. IOS Press (2014). https://doi.org/10.3233/978-1-61499-419-0-555
16. Manthey, N., Peñaloza, R., Rudolph, S.: Efficient axiom pinpointing in EL using sat technology. In: Description Logics (2016)
17. Peñaloza, R.: Axiom pinpointing. In: Cota, G., Daquino, M., Pozzato, G.L. (eds.) Applications and Practices in Ontology Design, Extraction, and Reasoning, Studies on the Semantic Web, vol. 49, pp. 162–177. IOS Press (2020). https://doi.org/10.3233/SSW200042
18. Penaloza, R., Sertkaya, B.: Understanding the complexity of axiom pinpointing in lightweight description logics. Artif. Intell. **250**, 80–104 (2017)

Choices, Invariance, Substitutions, and Formalizations

Sequent Calculi for Choice Logics

Michael Bernreiter[1](✉), Anela Lolic[1], Jan Maly[2], and Stefan Woltran[1]

[1] Institute of Logic and Computation, TU Wien, Vienna, Austria
{mbernrei,alolic,woltran}@dbai.tuwien.ac.at
[2] Institute for Logic, Language and Computation, University of Amsterdam, Amsterdam, The Netherlands
j.f.maly@uva.nl

Abstract. Choice logics constitute a family of propositional logics and are used for the representation of preferences, with especially *qualitative choice logic* (QCL) being an established formalism with numerous applications in artificial intelligence. While computational properties and applications of choice logics have been studied in the literature, only few results are known about the proof-theoretic aspects of their use. We propose a sound and complete sequent calculus for preferred model entailment in QCL, where a formula F is entailed by a QCL-theory T if F is true in all preferred models of T. The calculus is based on labeled sequent and refutation calculi, and can be easily adapted for different purposes. For instance, using the calculus as a cornerstone, calculi for other choice logics such as *conjunctive choice logic* (CCL) can be obtained in a straightforward way.

1 Introduction

Choice logics are propositional logics for the representation of alternative options for problem solutions [4]. These logics add new connectives to classical propositional logic that allow for the formalization of ranked options. A prominent example is *qualitative choice logic* (QCL for short) [7], which adds the connective *ordered disjunction* $\overrightarrow{\times}$ to classical propositional logic. Intuitively, $A \overrightarrow{\times} B$ means that if possible A, but if A is not possible than at least B. The semantics of a choice logic induce a preference ordering over the models of a formula.

As choice logics are well suited for preference handling, they have a multitude of applications in AI such as logic programming [8], alert correlation [3], or database querying [13]. But while computational properties and applications of choice logics have been studied in the literature, only few results are known about the proof-theoretic aspects of their use. In particular, there is no proof system capable of deriving valid sentences containing choice operators. In this paper we propose a sound and complete calculus for preferred model entailment in QCL that can easily be generalized to other choice logics.

Entailment in choice logics is non-monotonic: conclusions that have been drawn might not be derivable in light of new information. It is therefore not surprising that choice logics are related to other non-monotonic formalisms. For

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 331–349, 2022.
https://doi.org/10.1007/978-3-031-10769-6_20

instance, it is known [7] that QCL can capture propositional circumscription and that, if additional symbols in the language are admitted, circumscription can be used to generate models corresponding to the inclusion-preferred QCL models up to the additional atoms. We do not intend to use this translation of our choice logic formulas (or sequents) in order to employ an existing calculus for circumscription, for instance [5].

Instead, we define calculi in sequent format directly for choice logics, which are different from existing non-monotonic logics in the way non-monotonicity is introduced. Specifically, the non-standard part of our logics is a new logical connective which is fully embedded in the logical language. For this reason, calculi for choice logics also differ from most other calculi for non-monotonic logics: our calculi do not use non-standard inference rules as in default logic, modal operators expressing consistency or belief as in autoepistemic logic, or predicates whose extensions are minimized as in circumscription. However, one method that can also be applied to choice logics is the use of a refutation calculus (also known as rejection or antisequent calculus) axiomatising invalid formulas, i.e., non-theorems. Refutation calculi for non-monotonic logics were used in [5]. Specifically, by combining a refutation calculus with an appropriate sequent calculus, elegant proof systems for the central non-monotonic formalisms of default logic [16], autoepistemic logic [15], and circumscription [14] were obtained. However, to apply this idea to choice logics, we have to take another facet of their semantics into account.

With choice logics, we are working in a setting similar to many-valued logics. Interpretations ascribe a natural number called satisfaction degree to choice logic formulas. Preferred models of a formula are then those models with the least degree. There are several kinds of sequent calculus systems for many-valued logics, where the representation as a hypersequent calculus [1,10] plays a prominent role. However, there are crucial differences between choice logics and many-valued logics in the usual sense. Firstly, choice logic interpretations are classical, i.e., they set propositional variables to either true or false. Secondly, non-classical satisfaction degrees only arise when choice connectives, e.g. ordered disjunction in QCL, occur in a formula. Thirdly, when applying a choice connective $\circ$ to two formulas A and B, the degree of $A \circ B$ does not only depend on the degrees of A and B, but also on the maximum degrees that A and B can possibly assume. Therefore, techniques used in proof systems for conventional many-valued logics can not be applied directly to choice logics.

In [11] a sequent calculus based system for reasoning with contrary-to-duty obligations was introduced, where a non-classical connective was defined to capture the notion of reparational obligation, which is in force only when a violation of a norm occurs. This is related to the ordered disjunction in QCL, however, based on the intended use in [11] the system was defined only for the occurrence of the new connective on the right side of the sequent sign. We aim for a proof system for reasoning with choice logic operators, and to deduce formulas from choice logic formulas. Thus, we need a calculus with left and right inference rules.

To obtain such a calculus we combine the idea of a refutation calculus with methods developed for multi-valued logics in a novel way. First, we develop a (monotonic) sequent calculus for reasoning about satisfaction degrees using a labeled calculus, a method developed for (finite) many-valued logics [2,9,12]. Secondly, we define a labeled refutation calculus for reasoning about invalidity in terms of satisfaction degrees. Finally, we join both calculi to obtain a sequent calculus for the non-monotonic entailment of QCL. To this end, we introduce a new, non-monotonic inference rule that has sequents of the two labeled calculi as premises and formalizes degree minimization.

The rest of this paper is organized as follows. In the next section we present the basic notions of choice logics and introduce the most prominent choice logics QCL and CCL (*conjunctive choice logic*). In Sect. 3 we develop a labeled sequent calculus for propositional logic extended by the QCL connective $\vec{\times}$. This calculus is shown to be sound and complete and already can be used to derive interesting sentences containing choice operators. In Sect. 4 we extend the previously defined sequent calculus with an appropriate refutation calculus and non-monotonic reasoning, to capture entailment in QCL. The developed methodology for QCL can be extended to other choice logics as well. In particular we show in Sect. 5 how the calculi can be adapted for CCL.

2 Choice Logics

First, we formally define the notion of choice logics in accordance with the choice logic framework of [4] before giving concrete examples in the form of QCL and CCL. Finally, we define preferred model entailment.

Definition 1. *Let $\mathcal{U}$ denote the alphabet of propositional variables. The set of choice connectives $\mathcal{C}_\mathcal{L}$ of a choice logic $\mathcal{L}$ is a finite set of symbols such that $\mathcal{C}_\mathcal{L} \cap \{\neg, \wedge, \vee\} = \emptyset$. The set $\mathcal{F}_\mathcal{L}$ of formulas of $\mathcal{L}$ is defined inductively as follows: (i) $a \in \mathcal{F}_\mathcal{L}$ for all $a \in \mathcal{U}$; (ii) if $F \in \mathcal{F}_\mathcal{L}$, then $(\neg F) \in \mathcal{F}_\mathcal{L}$; (iii) if $F, G \in \mathcal{F}_\mathcal{L}$, then $(F \circ G) \in \mathcal{F}_\mathcal{L}$ for $\circ \in (\{\wedge, \vee\} \cup \mathcal{C}_\mathcal{L})$.*

For example, $\mathcal{C}_{\mathrm{QCL}} = \{\vec{\times}\}$ and $((a \vec{\times} c) \wedge (b \vec{\times} c)) \in \mathcal{F}_{\mathrm{QCL}}$. Formulas that do not contain a choice connective are referred to as classical formulas.

The semantics of a choice logic is given by two functions, satisfaction degree and optionality. The satisfaction degree of a formula given an interpretation is either a natural number or ∞. The lower this degree, the more preferable the interpretation. The optionality of a formula describes the maximum finite satisfaction degree that this formula can be ascribed, and is used to penalize non-satisfaction.

Definition 2. *The optionality of a choice connective $\circ \in \mathcal{C}_\mathcal{L}$ in a choice logic $\mathcal{L}$ is given by a function $opt^\circ_\mathcal{L} \colon \mathbb{N}^2 \to \mathbb{N}$ such that $opt^\circ_\mathcal{L}(k, \ell) \leq (k+1) \cdot (\ell + 1)$ for all $k, \ell \in \mathbb{N}$. The optionality of an $\mathcal{L}$-formula is given via $opt_\mathcal{L} \colon \mathcal{F}_\mathcal{L} \to \mathbb{N}$ with (i) $opt_\mathcal{L}(a) = 1$ for every $a \in \mathcal{U}$; (ii) $opt_\mathcal{L}(\neg F) = 1$; (iii) $opt_\mathcal{L}(F \wedge G) = opt_\mathcal{L}(F \vee G) = max(opt_\mathcal{L}(F), opt_\mathcal{L}(G))$; (iv) $opt_\mathcal{L}(F \circ G) = opt^\circ_\mathcal{L}(opt_\mathcal{L}(F), opt_\mathcal{L}(G))$ for every choice connective $\circ \in \mathcal{C}_\mathcal{L}$.*

The optionality of a classical formula is always 1. Note that, for any choice connective $\circ$, the optionality of $F \circ G$ is bounded such that $opt_{\mathcal{L}}(F \circ G) \leq (opt_{\mathcal{L}}(F) + 1) \cdot (opt_{\mathcal{L}}(G) + 1)$. In the following, we write $\overline{\mathbb{N}}$ for $(\mathbb{N} \cup \{\infty\})$.

Definition 3. *The satisfaction degree of a choice connective $\circ \in \mathcal{C}_{\mathcal{L}}$ in a choice logic $\mathcal{L}$ is given by a function $deg^{\circ}_{\mathcal{L}} \colon \mathbb{N}^2 \times \overline{\mathbb{N}}^2 \to \overline{\mathbb{N}}$ such that $deg^{\circ}_{\mathcal{L}}(k, \ell, m, n) \leq opt^{\circ}_{\mathcal{L}}(k, \ell)$ or $deg^{\circ}_{\mathcal{L}}(k, \ell, m, n) = \infty$ for all $k, \ell \in \mathbb{N}$ and all $m, n \in \overline{\mathbb{N}}$. The satisfaction degree of an $\mathcal{L}$-formula under an interpretation $\mathcal{I} \subseteq \mathcal{U}$ is given via the function $deg_{\mathcal{L}} \colon 2^{\mathcal{U}} \times \mathcal{F}_{\mathcal{L}} \to \overline{\mathbb{N}}$ with*

1. *$deg_{\mathcal{L}}(\mathcal{I}, a) = 1$ if $a \in \mathcal{I}$, $deg_{\mathcal{L}}(\mathcal{I}, a) = \infty$ otherwise for every $a \in \mathcal{U}$;*
2. *$deg_{\mathcal{L}}(\mathcal{I}, \neg F) = 1$ if $deg_{\mathcal{L}}(\mathcal{I}, F) = \infty$, $deg_{\mathcal{L}}(\mathcal{I}, \neg F) = \infty$ otherwise;*
3. *$deg_{\mathcal{L}}(\mathcal{I}, F \wedge G) = max(deg_{\mathcal{L}}(\mathcal{I}, F), deg_{\mathcal{L}}(\mathcal{I}, G))$;*
4. *$deg_{\mathcal{L}}(\mathcal{I}, F \vee G) = min(deg_{\mathcal{L}}(\mathcal{I}, F), deg_{\mathcal{L}}(\mathcal{I}, G))$;*
5. *$deg_{\mathcal{L}}(\mathcal{I}, F \circ G) = deg^{\circ}_{\mathcal{L}}(opt_{\mathcal{L}}(F), opt_{\mathcal{L}}(G), deg_{\mathcal{L}}(\mathcal{I}, F), deg_{\mathcal{L}}(\mathcal{I}, G))$, $\circ \in \mathcal{C}_{\mathcal{L}}$.*

We also write $\mathcal{I} \models^{\mathcal{L}}_{m} F$ for $deg_{\mathcal{L}}(\mathcal{I}, F) = m$. If $m < \infty$, we say that $\mathcal{I}$ satisfies F (to a finite degree), and if $m = \infty$, then $\mathcal{I}$ does not satisfy F. If F is a classical formula, then $\mathcal{I} \models^{\mathcal{L}}_{1} F \iff \mathcal{I} \models F$ and $\mathcal{I} \models^{\mathcal{L}}_{\infty} F \iff \mathcal{I} \not\models F$. The symbols $\top$ and $\bot$ are shorthand for the formulas $(a \vee \neg a)$ and $(a \wedge \neg a)$, where a can be any variable. We have $opt_{\mathcal{L}}(\top) = opt_{\mathcal{L}}(\bot) = 1$, $deg_{\mathcal{L}}(\mathcal{I}, \top) = 1$ and $deg_{\mathcal{L}}(\mathcal{I}, \bot) = \infty$ for any interpretation $\mathcal{I}$ in every choice logic.

Models and preferred models of formulas are defined in the following way:

Definition 4. *Let $\mathcal{L}$ be a choice logic, $\mathcal{I}$ an interpretation, and F an $\mathcal{L}$-formula. $\mathcal{I}$ is a model of F, written as $\mathcal{I} \in Mod_{\mathcal{L}}(F)$, if $deg_{\mathcal{L}}(\mathcal{I}, F) < \infty$. $\mathcal{I}$ is a preferred model of F, written as $\mathcal{I} \in Prf_{\mathcal{L}}(F)$, if $\mathcal{I} \in Mod_{\mathcal{L}}(F)$ and $deg_{\mathcal{L}}(\mathcal{I}, F) \leq deg_{\mathcal{L}}(\mathcal{J}, F)$ for all other interpretations $\mathcal{J}$.*

Moreover, we define the notion of classical counterparts for choice connectives.

Definition 5. *Let $\mathcal{L}$ be a choice logic. The classical counterpart of a choice connective $\circ \in \mathcal{C}_{\mathcal{L}}$ is the classical binary connective $\circledast$ such that, for all atoms a and b, $deg_{\mathcal{L}}(\mathcal{I}, a \circ b) < \infty \iff \mathcal{I} \models a \circledast b$. The classical counterpart of an $\mathcal{L}$-formula F is denoted as $cp(F)$ and is obtained by replacing all occurrences of choice connectives in F by their classical counterparts.*

A natural property of known choice logics is that choice connectives can be replaced by their classical counterpart without affecting satisfiability, meaning that $deg_{\mathcal{L}}(\mathcal{I}, F) < \infty \iff \mathcal{I} \models cp(F)$ holds for all $\mathcal{L}$-formulas F.

So far we introduced choice logics in a quite abstract way. We now introduce two particular instantiations, namely QCL, the first and most prominent choice logic in the literature, and CCL, which introduces a connective $\overrightarrow{\odot}$ called ordered conjunction in place of QCL's ordered disjunction.

Definition 6. QCL *is the choice logic such that* $\mathcal{C}_{\text{QCL}} = \{\vec{\times}\}$*, and, if* $k = opt_{\text{QCL}}(F)$*,* $\ell = opt_{\text{QCL}}(G)$*,* $m = deg_{\text{QCL}}(\mathcal{I}, F)$*, and* $n = deg_{\text{QCL}}(\mathcal{I}, G)$*, then*

$$opt_{\text{QCL}}(F \vec{\times} G) = opt^{\vec{\times}}_{\text{QCL}}(k, \ell) = k + \ell, \ \textit{and}$$

$$deg_{\text{QCL}}(\mathcal{I}, F \vec{\times} G) = deg^{\vec{\times}}_{\text{QCL}}(k, \ell, m, n) = \begin{cases} m & \textit{if } m < \infty; \\ n + k & \textit{if } m = \infty, n < \infty; \\ \infty & \textit{otherwise.} \end{cases}$$

In the above definition, we can see how optionality is used to penalize non-satisfaction: given a QCL-formula $F \vec{\times} G$ and an interpretation $\mathcal{I}$, if $\mathcal{I}$ satisfies F (to some finite degree), then $deg_{\text{QCL}}(\mathcal{I}, F \vec{\times} G) = deg_{\text{QCL}}(\mathcal{I}, F)$; if $\mathcal{I}$ does not satisfy F, then $deg_{\text{QCL}}(\mathcal{I}, F \vec{\times} G) = opt_{\text{QCL}}(F) + deg_{\text{QCL}}(\mathcal{I}, G)$. Since $deg_{\text{QCL}}(\mathcal{I}, F) \leq opt_{\text{QCL}}(F)$, interpretations that satisfy F result in a lower degree, i.e., are more preferable, compared to interpretations that do not satisfy F. Let us take a look at a concrete example:

Example 1. Consider the QCL-formula $F = (a \vec{\times} c) \wedge (b \vec{\times} c)$. Note that the classical counterpart of $\vec{\times}$ is $\vee$, i.e., $cp(F) = (a \vee c) \wedge (b \vee c)$. Thus, $\{c\}, \{a, b\}, \{a, c\}$, $\{b, c\}, \{a, b, c\} \in Mod_{\text{QCL}}(F)$. Of these models, $\{a, b\}$ and $\{a, b, c\}$ satisfy F to a degree of 1 while $\{c\}$, $\{a, c\}$, and $\{b, c\}$ satisfy F to a degree of 2. Therefore, $\{a, b\}, \{a, b, c\} \in Prf_{\text{QCL}}(F)$.

Next, we define CCL. Note that we follow the revised definition of CCL [4], which differs from the initial specification[1]. Intuitively, given a CCL-formula $F \vec{\odot} G$ it is best to satisfy both F and G, but also acceptable to satisfy only F.

Definition 7. CCL *is the choice logic such that* $\mathcal{C}_{\text{CCL}} = \{\vec{\odot}\}$*, and, if* $k = opt_{\text{CCL}}(F)$*,* $\ell = opt_{\text{CCL}}(G)$*,* $m = deg_{\text{CCL}}(\mathcal{I}, F)$*, and* $n = deg_{\text{CCL}}(\mathcal{I}, G)$*, then*

$$opt_{\text{CCL}}(F \vec{\odot} G) = k + \ell, \ \textit{and}$$

$$deg_{\text{CCL}}(\mathcal{I}, F \vec{\odot} G) = \begin{cases} n & \textit{if } m = 1, n < \infty; \\ m + \ell & \textit{if } m < \infty \ \textit{and} \ (m > 1 \ \textit{or} \ n = \infty); \\ \infty & \textit{otherwise.} \end{cases}$$

Example 2. Consider the CCL-formula $G = (a \vec{\odot} c) \wedge (b \vec{\odot} c)$. Note that the classical counterpart of $\vec{\odot}$ is the first projection, i.e., $cp(G) = a \wedge b$. Thus, $\{a, b\}$, $\{a, b, c\} \in Mod_{\text{CCL}}(G)$. Of these models, $\{a, b, c\}$ satisfies G to a degree of 1 while $\{a, b\}$ satisfies G to a degree of 2. Therefore, $\{a, b, c\} \in Prf_{\text{CCL}}(G)$.

If $\mathcal{L}$ is a choice logic, then a set of $\mathcal{L}$-formulas is called an $\mathcal{L}$-theory. An $\mathcal{L}$-theory T entails a classical formula F, written as $T \mid\!\sim F$, if F is true in all preferred models of T. However, we first need to define what the preferred models of a choice logic theory are. There are several approaches for this. In the original QCL paper [7], a lexicographic and an inclusion-based approach were introduced.

[1] It seems that, under the initial definition of CCL, $a \vec{\odot} b$ is always ascribed a degree of 1 or ∞, i.e., non-classical degrees can not be obtained (cf. Definition 8 in [6]).

Definition 8. *Let $\mathcal{L}$ be a choice logic, $\mathcal{I}$ an interpretation, and T an $\mathcal{L}$-theory. $\mathcal{I} \in Mod_{\mathcal{L}}(T)$ if $deg_{\mathcal{L}}(\mathcal{I}, F) < \infty$ for all $F \in T$. $\mathcal{I}^k_{\mathcal{L}}(T)$ denotes the set of formulas in T satisfied to a degree of k by $\mathcal{I}$, i.e., $\mathcal{I}^k_{\mathcal{L}}(T) = \{F \in T \mid deg_{\mathcal{L}}(\mathcal{I}, F) = k\}$.*

- *$\mathcal{I}$ is a lexicographically preferred model of T, written as $\mathcal{I} \in Prf^{lex}_{\mathcal{L}}(T)$, if $\mathcal{I} \in Mod_{\mathcal{L}}(T)$ and if there is no $\mathcal{J} \in Mod_{\mathcal{L}}(T)$ such that, for some $k \in \mathbb{N}$ and all $l < k$, $|\mathcal{I}^k_{\mathcal{L}}(T)| < |\mathcal{J}^k_{\mathcal{L}}(T)|$ and $|\mathcal{I}^l_{\mathcal{L}}(T)| = |\mathcal{J}^l_{\mathcal{L}}(T)|$ holds.*
- *$\mathcal{I}$ is an inclusion-based preferred model of T, written as $\mathcal{I} \in Prf^{inc}_{\mathcal{L}}(T)$, if $\mathcal{I} \in Mod_{\mathcal{L}}(T)$ and if there is no $\mathcal{J} \in Mod_{\mathcal{L}}(T)$ such that, for some $k \in \mathbb{N}$ and all $l < k$, $\mathcal{I}^k_{\mathcal{L}}(T) \subset \mathcal{J}^k_{\mathcal{L}}(T)$ and $\mathcal{I}^l_{\mathcal{L}}(T) = \mathcal{J}^l_{\mathcal{L}}(T)$ holds.*

In our calculus for preferred model entailment we focus on the lexicographic approach, but it will become clear how it can be adapted to other preferred model semantics (see Sect. 4). We now formally define preferred model entailment:

Definition 9. *Let $\mathcal{L}$ be a choice logic, T an $\mathcal{L}$-theory, S a classical theory, and $\sigma \in \{lex, inc\}$. $T \mathrel{\mid\!\sim}^{\sigma}_{\mathcal{L}} S$ if for all $\mathcal{I} \in Prf^{\sigma}_{\mathcal{L}}(T)$ there is $F \in S$ such that $\mathcal{I} \models F$.*

Example 3. Consider the QCL-theory $T = \{\neg(a \wedge b), a \vec{\times} c, b \vec{\times} c\}$. Then $\{c\}, \{a, c\}, \{b, c\} \in Mod_{\text{QCL}}(T)$. Note that, because of $\neg(a \wedge b)$, a model of T can not satisfy both $a \vec{\times} c$ and $b \vec{\times} c$ to a degree of 1. Specifically,

$$\begin{aligned}
&\{a, c\}^1_{\text{QCL}}(T) = \{\neg(a \wedge b), a \vec{\times} c\} \text{ and } \{a, c\}^2_{\text{QCL}}(T) = \{b \vec{\times} c\},\\
&\{b, c\}^1_{\text{QCL}}(T) = \{\neg(a \wedge b), b \vec{\times} c\} \text{ and } \{b, c\}^2_{\text{QCL}}(T) = \{a \vec{\times} c\},\\
&\{c\}^1_{\text{QCL}}(T) = \{\neg(a \wedge b)\} \text{ and } \{c\}^2_{\text{QCL}}(T) = \{a \vec{\times} c, b \vec{\times} c\}.
\end{aligned}$$

Thus, $\{a, c\}, \{b, c\} \in Prf^{lex}_{\text{QCL}}(T)$ but $\{c\} \notin Prf^{lex}_{\text{QCL}}(T)$. It can be concluded that $T \mathrel{\mid\!\sim}^{lex}_{\text{QCL}} c \wedge (a \vee b)$. However, $T \mathrel{\not\mid\!\sim}^{lex}_{\text{QCL}} a$ and $T \mathrel{\not\mid\!\sim}^{lex}_{\text{QCL}} b$.

It is easy to see that preferred model entailment is non-monotonic. For example, $\{a \vec{\times} b\} \mathrel{\mid\!\sim}^{lex}_{\text{QCL}} a$ but $\{a \vec{\times} b, \neg a\} \mathrel{\not\mid\!\sim}^{lex}_{\text{QCL}} a$.

3 The Sequent Calculus L[QCL]

As a first step towards a calculus for preferred model entailment, we propose a labeled calculus [2,12] for reasoning about the satisfaction degrees of QCL formulas in sequent format and prove its soundness and completeness. One advantage of the sequent calculus format is having symmetrical left and right rules for all connectives, in particular for the choice connectives. This is in contrast to the representation of ordered disjunction in the calculus for deontic logic [11], in which only right-hand side rules are considered.

As the calculus will be concerned with satisfaction degrees rather than preferred models, we need to define entailment in terms of satisfaction degrees. To this end, the formulas occurring in the sequents of our calculus are labeled with natural numbers, i.e., they are of the form $(A)_k$, where A is a choice logic formula and $k \in \mathbb{N}$. $(A)_k$ is satisfied by those interpretations that satisfy A to a degree of

k. Instead of labeling formulas with degree ∞ we use the negated formula, i.e., instead of $(A)_\infty$ we use $(\neg A)_1$. We observe that $(A)_k$ for $opt_{\mathcal{L}}(A) > k$ can never have a model. We will deal with such formulas by replacing them with $(\bot)_1$. For classical formulas, we may write A for $(A)_1$.

Definition 10. *Let $(A_1)_{k_1}, \dots, (A_m)_{k_m}$ and $(B_1)_{l_1}, \dots, (B_n)_{l_n}$ be labeled* QCL*-formulas. $(A_1)_{k_1}, \dots, (A_m)_{k_m} \vdash (B_1)_{l_1}, \dots, (B_n)_{l_n}$ is a labeled* QCL*-sequent. $\Gamma \vdash \Delta$ is valid iff every interpretation that satisfies all labeled formulas in Γ to the degree specified by the label also satisfies at least one labeled formula in Δ to the degree specified by the label.*

Note that entailment in terms of satisfaction degrees, as defined above, is montonic. Frequently we will write $(A)_{<k}$ as shorthand for $(A)_1, \dots, (A)_{k-1}$ and $(A)_{>k}$ for $(A)_{k+1}, \dots, (A)_{opt_{\text{QCL}}(A)}, (\neg A)_1$. Moreover, $\langle \Gamma, (A)_i \vdash \Delta \rangle_{i<k}$ denotes the sequence of sequents

$$\Gamma, (A)_1 \vdash \Delta \dots \Gamma, (A)_{k-1} \vdash \Delta.$$

Analogously, $\langle \Gamma, (A)_i \vdash \Delta \rangle_{i>k}$ stands for the sequence of sequents $\Gamma, (A)_{k+1} \vdash \Delta \dots \Gamma, (A)_{opt_{\text{QCL}}(A)} \vdash \Delta \quad \Gamma, (\neg A)_1 \vdash \Delta$.

We define the sequent calculus **L**[QCL] over labeled sequents below. In addition to introducing inference rules for $\vec{\times}$ we have to modify the inference rules for conjunction and disjunction of propositional **LK**. The idea behind the $\vee$-left rule is that a model M of $(A)_k$ is only a model of $(A \vee B)_k$ if there is no $l < k$ s.t. M is a model of $(B)_l$. Therefore, every model of $(A \vee B)_k$ is a model of Δ iff

- every model of $(A)_k$ is a model of Δ or of some $(B)_l$ with $l < k$,
- every model of $(B)_k$ is a model of Δ or of some $(A)_l$ with $l < k$.

Essentially the same idea works for $\wedge$-left but with $l > k$. For the $\vee$-right rule, in order for every model of Γ to be a model of $(A \vee B)_k$, every model of Γ must either be a model of $(A)_k$ or of $(B)_k$ and no model of Γ can be a model of $(A)_l$ for $l < k$, i.e., $\Gamma, (A)_l \vdash \bot$. Similarly for $\wedge$-right.

Definition 11 (L[QCL]). *The axioms of* **L**[QCL] *are of the form $(p)_1 \vdash (p)_1$ for propositional variables p. The inference rules are given below. For the structural and logical rules, whenever a labeled formula $(F)_k$ appears in the conclusion of an inference rule it holds that $k \leq opt_{\mathcal{L}}(F)$.*

The structural rules are:

$$\frac{\Gamma \vdash \Delta}{\Gamma, (A)_k \vdash \Delta}\,wl \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash (A)_k, \Delta}\,wr \qquad \frac{\Gamma, (A)_k, (A)_k \vdash \Delta}{\Gamma, (A)_k \vdash \Delta}\,cl \qquad \frac{\Gamma \vdash (A)_k, (A)_k, \Delta}{\Gamma \vdash (A)_k, \Delta}\,cr$$

The logical rules are:

$$\frac{\Gamma \vdash (cp(A))_1, \Delta}{\Gamma, (\neg A)_1 \vdash \Delta}\,\neg l \qquad \frac{\Gamma, (cp(A))_1 \vdash \Delta}{\Gamma \vdash (\neg A)_1, \Delta}\,\neg r$$

$$\frac{\Gamma, (A)_k \vdash (B)_{<k}, \Delta \qquad \Gamma, (B)_k \vdash (A)_{<k}, \Delta}{\Gamma, (A \vee B)_k \vdash \Delta}\,\vee l \quad \frac{\langle \Gamma, (A)_i \vdash \Delta \rangle_{i<k} \quad \langle \Gamma, (B)_i \vdash \Delta \rangle_{i<k} \quad \Gamma \vdash (A)_k, (B)_k, \Delta}{\Gamma \vdash (A \vee B)_k, \Delta}\,\vee r$$

$$\frac{\Gamma, (A)_k \vdash (B)_{>k}, \Delta \qquad \Gamma, (B)_k \vdash (A)_{>k}, \Delta}{\Gamma, (A \wedge B)_k \vdash \Delta} \wedge l \qquad \frac{\langle \Gamma, (A)_i \vdash \Delta \rangle_{i>k} \quad \langle \Gamma, (B)_i \vdash \Delta \rangle_{i>k} \quad \Gamma \vdash (A)_k, (B)_k, \Delta}{\Gamma \vdash (A \wedge B)_k, \Delta} \wedge r$$

The rules for ordered disjunction, with $k \leq opt_{\mathcal{L}}(A)$ and $l \leq opt_{\mathcal{L}}(B)$, are:

$$\frac{\Gamma, (A)_k \vdash \Delta}{\Gamma, (A \vec{\times} B)_k \vdash \Delta} \vec{\times} l_1 \qquad \frac{\Gamma, (B)_l, (\neg A)_1 \vdash \Delta}{\Gamma, (A \vec{\times} B)_{opt_{\mathrm{QCL}}(A)+l} \vdash \Delta} \vec{\times} l_2$$

$$\frac{\Gamma \vdash (A)_k, \Delta}{\Gamma \vdash (A \vec{\times} B)_k, \Delta} \vec{\times} r_1 \qquad \frac{\Gamma \vdash (\neg A)_1, \Delta \qquad \Gamma \vdash (B)_l, \Delta}{\Gamma \vdash (A \vec{\times} B)_{opt_{\mathrm{QCL}}(A)+l}, \Delta} \vec{\times} r_2$$

The degree overflow rules[2]*, with $k \in \mathbb{N}$, are:*

$$\frac{\Gamma, \bot \vdash \Delta}{\Gamma, (A)_{opt_{\mathrm{QCL}}(A)+k} \vdash \Delta} \, dol \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash (A)_{opt_{\mathrm{QCL}}(A)+k}, \Delta} \, dor$$

Observe that the modified $\wedge$ and $\vee$ inference rules correspond to the $\wedge$ and $\vee$ inference rules of propositional **LK** in case we are dealing only with classical formulas. Our $\wedge$-left rule splits the proof-tree unnecessarily for classical theories, and the $\wedge$-right rule adds an unnecessary third condition $\Gamma \vdash A, B, \Delta$. These additional conditions are necessary when dealing with non-classical formulas.

The intuition behind the degree overflow rules is that we sometimes need to fix invalid sequences, i.e., sequences in which a formula F is assigned a label k with $opt_{\mathrm{QCL}}(F) < k < \infty$.

Example 4. The following is an **L**[QCL]-proof of a valid sequent.[3]

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots \\ b \vee c, \neg a, b \vdash a \wedge b, a \wedge c, b\end{array}}{b \vee c, (a \vec{\times} b)_2 \vdash a \wedge b, a \wedge c, b} \vec{\times} l_2}{(a \vec{\times} b)_2 \vdash \neg(b \vec{\times} c), a \wedge b, a \wedge c, b} \neg r \qquad \dfrac{\dfrac{\begin{array}{c}\vdots \\ a \vee b, \neg b, c \vdash a \wedge b, a \wedge c, b\end{array}}{a \vee b, (b \vec{\times} c)_2 \vdash a \wedge b, a \wedge c, b} \vec{\times} l_2}{(b \vec{\times} c)_2 \vdash \neg(a \vec{\times} b), a \wedge b, a \wedge c, b} \neg r}{((a \vec{\times} b) \wedge (b \vec{\times} c))_2 \vdash a \wedge b, a \wedge c, b} \wedge l}{\neg(a \wedge b), ((a \vec{\times} b) \wedge (b \vec{\times} c))_2 \vdash a \wedge c, b} \neg l$$

Example 5. The following proof shows how the $\wedge r$-rule can introduce more than three premises. Note that we make use of the *dol*-rule in the leftmost branch.

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ a, b, \bot \vdash\end{array}}{a, b, (a)_2 \vdash} dol \quad \begin{array}{c}\vdots \\ a, b, \neg a \vdash\end{array} \quad \dfrac{\begin{array}{c}\vdots \\ a, b, c, \neg b \vdash\end{array}}{a, b, (b \vec{\times} c)_2 \vdash} \vec{\times} l_2 \quad \dfrac{\begin{array}{c}\vdots \\ a, b \vdash b \vee c\end{array}}{a, b, \neg(b \vec{\times} c) \vdash} \neg l \quad \dfrac{\begin{array}{c}\vdots \\ a, b \vdash a, b\end{array}}{a, b \vdash a, (b \vec{\times} c)_1} \vec{\times} r_1}{a, b \vdash (a \wedge (b \vec{\times} c))_1} \wedge r$$

We now show soundness and completeness of **L**[QCL].

[2] *dol*/*dor* stands for degree overflow left/right.

[3] Note that, once we reach sequents containing only classical formulas, we do not continue the proof. However, it can be verified that the classical sequents on the left and right branch are provable in this case. Moreover, given a formula $(A)_1$ with a label of 1, the label is often omitted for readability.

Proposition 1. $\mathbf{L}[\mathrm{QCL}]$ *is sound.*

Proof. We show for all rules that they are sound.

- For (ax) and the structural rules this is clearly the case.
- $(\neg r)$ and $(\neg l)$: follows from the fact that $deg_{\mathrm{QCL}}(\mathcal{I}, F) < \infty \iff \mathcal{I} \models cp(F)$ for all QCL-formulas F.
- $(\vee l)$: Assume that the conclusion of the rule is not valid, i.e., there is a model M of Γ and $(A \vee B)_k$ that is not a model of Δ. Then, M satisfies either A or B to degree k and neither to a degree smaller than k. Assume M satisfies A to a degree of k, the other case is symmetric. Then M is a model of Γ and $(A)_k$ but, by assumption, neither of Δ nor of $(B)_j$ for any $j < k$. Hence at least one of the premises is not valid. Analogously for $(\wedge l)$.
- $(\vee r)$: Assume there is a model M of Γ that is not a model of Δ or of $(A \vee B)_k$. There are two possible cases why M is not a model of $(A \vee B)_k$: (1) M satisfies neither A nor B to degree k. But then the premise $\Gamma \vdash (A)_k, (B)_k, \Delta$ is not valid as M is also not a model of Δ by assumption. (2) M satisfies either A or B to a degree smaller than k. Assume that M satisfies A to degree $j < k$ (the other case is symmetric). Then the premise $\Gamma, (A)_j \vdash \Delta$ is not valid. Indeed, M is a model of Γ and $(A)_j$ but not of Δ. Analogously for $(\wedge r)$.
- $(\overrightarrow{\times} l_1)$ and $(\overrightarrow{\times} r_1)$: follows from the fact that $(A)_k$ has the same models as $(A \overrightarrow{\times} B)_k$ for $k \leq opt_{\mathcal{L}}(A)$.
- $(\overrightarrow{\times} l_2)$: Assume the conclusion of the rule is not valid and let M be the model witnessing this. Then M is a model of $(A \overrightarrow{\times} B)_{opt_{QCL}(A)+l}$. By definition, M satisfies B to degree l and is not a model of A. However, then it is also a model of Γ, $(B)_l$ and $(\neg A)_1$, which means that the premise is not valid.
- $(\overrightarrow{\times} r_2)$. Assume that both premises are valid, i.e., every model of Γ is either a model of Δ or of $(\neg A)_1$ and $(B)_l$ with $l \leq opt_{\mathcal{L}}(B)$. Now, by definition, any model that is not a model of A (and hence a model of $(\neg A)_1$) and of $(B)_l$ satisfies $A \overrightarrow{\times} B$ to degree $opt_{QCL}(A) + l$. Therefore, every model of Γ is either a model of Δ or of $(A \overrightarrow{\times} B)_{opt_{QCL}(A)+l}$, which means that the conclusion of the rule is valid.
- (dol): $\Gamma, \bot$ has no models, i.e., the premise $\Gamma, \bot \vdash \Delta$ is valid. Crucially, the sequent $\Gamma, (A)_{opt_{\mathrm{QCL}}(A)+k}$ has no models as well since A cannot be satisfied to a degree m with $opt_{\mathcal{L}}(A) < m < \infty$. (dor) is clearly sound. □

Proposition 2. $\mathbf{L}[\mathrm{QCL}]$ *is complete.*

Proof. We show this by induction over the (aggregated) formula complexity of the non-classical formulas.

- For the base case, we observed that if all formulas are classical and labeled with 1, then all our rules reduce to the classical sequent calculus, which is known to be complete. Moreover, we observe that $(A)_1$ is equivalent to A. Hence, we can turn labeled atoms into classical atoms.
- Assume that a sequent of the form $\Gamma, (A)_{opt_{\mathrm{QCL}}(A)+k} \vdash \Delta$ with $k \in \mathbb{N}$ is valid. Since $\Gamma, \bot$ has no models, $\Gamma, \bot \vdash \Delta$ is valid and, by the induction hypothesis, provable. Thus, $\Gamma, (A)_{opt_{\mathrm{QCL}}(A)+k} \vdash \Delta$ is provable using the (dol) rule.

- Assume that a sequent $\Gamma \vdash (A)_{opt_{\mathrm{QCL}}(A)+k}, \Delta$ is valid. $(A)_{opt_{\mathrm{QCL}}(A)+k}$ can not be satisfied, i.e., $\Gamma \vdash \Delta$ is valid and, by the induction hypothesis, provable. Therefore, $\Gamma \vdash (A)_{opt_{\mathrm{QCL}}(A)+k}, \Delta$ is provable using the (dor) rule.
- Assume that a sequent of the form $\Gamma \vdash (\neg A)_1, \Delta$ is valid. Then every model of Γ is either a model of $(\neg A)_1$ or of Δ. In other words, every model of Γ that is not a model of $(\neg A)_1$ (i.e., is model of $cp(A)$) is a model of Δ. Therefore, every interpretation that is a model of both Γ and $cp(A)$ must be a model of Δ. It follows that $\Gamma, cp(A) \vdash \Delta$ is valid and, by the induction hypothesis, provable. Thus, $\Gamma \vdash (\neg A)_1, \Delta$ is provable using the $(\neg r)$ rule. Similarly for $\Gamma, (\neg A)_1 \vdash \Delta$.
- Assume that a sequent of the form $\Gamma, (A \vee B)_k \vdash \Delta$ is valid, with $k \leq opt_{\mathcal{L}}(A \vee B)$. We claim that then both $\Gamma, (A)_k \vdash (B)_{<k}, \Delta$ and $\Gamma, (B)_k \vdash (A)_{<k}, \Delta$ are valid. Assume to the contrary that $\Gamma, (A)_k \vdash (B)_{<k}, \Delta$ is not valid (the other case is symmetric). Then, there is a model M of Γ and $(A)_k$ that is neither a model of $(B)_{<k}$ nor of Δ. But then M is also a model of Γ and $(A \vee B)_k$, but not of Δ, which contradicts the assumption that $\Gamma, (A \vee B)_k \vdash \Delta$ is valid. Therefore, both $\Gamma, (A)_k \vdash (B)_{<k}, \Delta$ and $\Gamma, (B)_k \vdash (A)_{<k}, \Delta$ are valid and, by the induction hypothesis, provable. This means that $\Gamma, (A \vee B)_k \vdash \Delta$ is provable by $(\vee l)$. Similarly for a sequent of the form $\Gamma, (A \wedge B)_k \vdash \Delta$.
- Assume that a sequent of the form $\Gamma \vdash (A \vee B)_k, \Delta$ is valid, with $k \leq opt_{\mathcal{L}}(A \vee B)$. We claim that then for all $i < k$ the sequents $\Gamma, (A)_i \vdash \Delta$ and $\Gamma, (B)_i \vdash \Delta$ and $\Gamma \vdash (A)_k, (B)_k, \Delta$ are valid. Assume by contradiction that there is an $i < k$ s.t. $\Gamma, (A)_i \vdash \Delta$ is not valid. Then, there is a model M of Γ and $(A)_i$ that is not a model of Δ. However, then M is a model of Γ but neither of Δ nor of $(A \vee B)_k$ (as M satisfies $A \vee B$ to degree $i \neq k$), which contradicts our assumption that $\Gamma \vdash (A \vee B)_k, \Delta$ is valid. The case that there is an $i < k$ s.t. $\Gamma, (B)_i \vdash \Delta$ is not valid is symmetric. Finally, we assume that $\Gamma \vdash (A)_k, (B)_k, \Delta$ is not valid. Then, there is a model M of Γ that is not a model of $(A)_k$, $(B)_k$ or Δ. Then, M is model of Γ but neither of Δ nor of $(A \vee B)_k$, contradicting our assumption. Therefore, all sequents listed above must be valid, and, by the induction hypothesis, $\Gamma \vdash (A \vee B)_k, \Delta$ is provable. Similarly for a sequent of the form $\Gamma \vdash (A \wedge B)_k, \Delta$.
- Assume that a sequent of the form $\Gamma, (A \overrightarrow{\times} B)_k \vdash \Delta$ with $k \leq opt_{\mathrm{QCL}}(A)$ is valid. Then $\Gamma, (A)_k \vdash \Delta$ is also valid since $(A \overrightarrow{\times} B)_k$ and $(A)_k$ have the same models if $k \leq opt_{\mathrm{QCL}}(A)$. By the induction hypothesis $\Gamma, (A \overrightarrow{\times} B)_k \vdash \Delta$ is provable. Analogously for sequents of the form $\Gamma \vdash (A \overrightarrow{\times} B)_k, \Delta$.
- Assume that a sequent of the form $\Gamma, (A \overrightarrow{\times} B)_{opt_{\mathrm{QCL}}(A)+l} \vdash \Delta$ is valid, with $l \leq opt_{\mathcal{L}}(B)$. We claim that the sequent $\Gamma, (B)_l, \neg A \vdash \Delta$ is then also valid. Indeed, if M is a model of Γ, $(B)_l$ and $\neg A$, then it is also a model of Γ and $(A \overrightarrow{\times} B)_{opt_{\mathrm{QCL}}(A)+l}$. Hence, by assumption, M must be a model of Δ. From this, we can conclude as before that $\Gamma, (A \overrightarrow{\times} B)_{opt_{\mathrm{QCL}}(A)+l} \vdash \Delta$ is provable.
- Assume that a sequent of the form $\Gamma \vdash (A \overrightarrow{\times} B)_{opt_{QCL}(A)+l}, \Delta$ is valid, with $l \leq opt_{\mathcal{L}}(B)$. We claim that then also the sequents $\Gamma \vdash \neg A, \Delta$ and $\Gamma \vdash (B)_l, \Delta$ are valid. Assume by contradiction that the first sequent is not valid. This means that there is a model M of Γ that is not a model of either $\neg A$

nor of Δ. However, then M is a model of A and therefore satisfies $A \vec{\times} B$ to a degree smaller than $opt_{\mathrm{QCL}}(A)$. This contradicts our assumption that $\Gamma \vdash (A \vec{\times} B)_{opt_{\mathrm{QCL}}(A)+l}, \Delta$ is valid. Assume now that the second sequent is not valid, i.e., that there is a model M of Γ that is neither a model of $(B)_l$ nor of Δ. Then, M cannot be a model of $(A \vec{\times} B)_{opt_{\mathrm{QCL}}(A)+l}$ and we again have a contradiction to our assumption. As before, it follows by the induction hypothesis that $\Gamma \vdash (A \vec{\times} B)_{opt_{\mathrm{QCL}}(A)+l}, \Delta$ is provable. □

So far we have not introduced a cut rule, and as we have shown our calculus is complete without such a rule. However, it is easy to see that we have cut-admissibility, i.e., **L**[QCL] can be extended by:

$$\frac{\Gamma \vdash (A)_k, \Delta \qquad \Gamma', (A)_k \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ cut$$

Another aspect of our calculus that should be mentioned is that, although **L**[QCL] is cut-free, we do not have the subformula property. This is especially obvious when looking at the rules for negation, where we use the classical counterpart $cp(A)$ of QCL-formulas. For example, $\neg(a \vec{\times} b)$ in the conclusion of the $\neg$-left rule becomes $cp(a \vec{\times} b) = a \vee b$ in the premise.

While we believe that **L**[QCL] is interesting in its own right, the question of how we can use it to obtain a calculus for preferred model entailment arises. Essentially, we have to add a rule that allows us to go from standard to preferred model inferences. As a first approach we consider theories $\Gamma \cup \{A\}$ with Γ consisting only of classical formulas and A being a QCL-formula. In this simple case, preferred models of $\Gamma \cup \{A\}$ are those models of $\Gamma \cup \{A\}$ that satisfy A to the smallest possible degree. One might add the following rule to **L**[QCL]:

$$\frac{\langle \Gamma, (A)_i \vdash \bot \rangle_{i<k} \qquad \Gamma, (A)_k \vdash \Delta}{\Gamma, A \mathrel{|\!\!\sim}^{lex}_{\mathrm{QCL}} \Delta}\ \mathrel{|\!\!\sim}_{naive}$$

Intuitively, the above rule states that, if there are no interpretations that satisfy Γ while also satisfying A to a degree lower than k, and if Δ follows from all models of $\Gamma, (A)_k$, then Δ is entailed by the preferred models of $\Gamma \cup \{A\}$. However, the obtained calculus **L**[QCL] + $\mathrel{|\!\!\sim}_{naive}$ derives invalid sequents.

Example 6. The invalid entailment $\neg a, a \vec{\times} b \mathrel{|\!\!\sim}^{lex}_{\mathrm{QCL}} a$ can be derived via $\mathrel{|\!\!\sim}_{naive}$.

$$\cfrac{\cfrac{\cfrac{a \vdash a}{\neg a, a \vdash a}\ wl}{\neg a, (a \vec{\times} b)_1 \vdash a}\ \vec{\times} l_1}{\neg a, a \vec{\times} b \mathrel{|\!\!\sim}^{lex}_{\mathrm{QCL}} a}\ \mathrel{|\!\!\sim}_{naive}$$

What is missing is an assertion that $\Gamma, (A)_k$ is satisfiable. Unfortunately, this can not be formulated in **L**[QCL]. A way of addressing this problem is to define a refutation calculus, as has been done for other non-monotonic logics [5].

4 Calculus for Preferred Model Entailment

We now introduce a calculus for preferred model entailment. However, as argued above, we first need to introduce the refutation calculus $\mathbf{L}[\mathrm{QCL}]^-$. In the literature, a rejection method for first-order logic with equality was first introduced in [17] and proved complete w.r.t. finite model theory. Our refutation calculus is based on a simpler rejection method for propositional logic defined in [5]. Using the refutation calculus, we prove that $(A)_k$ is satisfiable by deriving the antisequent $(A)_k \nvdash \bot$.

Definition 12. *A labeled* QCL*-antisequent is denoted by $\Gamma \nvdash \Delta$ and it is valid if and only if the corresponding labeled* QCL*-sequent $\Gamma \vdash \Delta$ is not valid, i.e., if at least one model that satisfies all formulas in Γ to the degree specified by the label satisfies no formula in Δ to the degree specified by the label.*

Below we give a definition of the refutation calculus $\mathbf{L}[\mathrm{QCL}]^-$. Note that most rules coincide with their counterparts in $\mathbf{L}[\mathrm{QCL}]$. Binary rules are translated into two rules; one inference rule per premise. $(\vee r)$ and $(\wedge l)$ in $\mathbf{L}[\mathrm{QCL}]$ have an unbounded number of premises, but due to their structure they can be translated into three inference rules. For $(\wedge r)$ we need to introduce two extra rules for the case that either A or B is not satisfied.

Definition 13 ($\mathbf{L}[\mathrm{QCL}]^-$). *The axioms of $\mathbf{L}[\mathrm{QCL}]^-$ are of the form $\Gamma \nvdash \Delta$, where Γ and Δ are disjoint sets of atoms and $\bot \notin \Gamma$. The inference rules of $\mathbf{L}[\mathrm{QCL}]^-$ are given below. Whenever a labeled formula $(F)_k$ appears in the conclusion of an inference rule it holds that $k \leq opt_{\mathcal{L}}(F)$.*

The logical rules are:

$$\frac{\Gamma, (cp(A))_1 \nvdash \Delta}{\Gamma \nvdash (\neg A)_1, \Delta}\nvdash \neg r \qquad \frac{\Gamma \nvdash (cp(A))_1, \Delta}{\Gamma, (\neg A)_1 \nvdash \Delta}\nvdash \neg l$$

$$\frac{\Gamma, (A)_k \nvdash (B)_{<k}, \Delta}{\Gamma, (A \vee B)_k \nvdash \Delta}\nvdash \vee l_1 \qquad \frac{\Gamma, (B)_k \nvdash (A)_{<k}, \Delta}{\Gamma, (A \vee B)_k \nvdash \Delta}\nvdash \vee l_2$$

$$\frac{\Gamma, (A)_i \nvdash \Delta}{\Gamma \nvdash (A \vee B)_k, \Delta}\nvdash \vee r_1 \qquad \frac{\Gamma, (B)_i \nvdash \Delta}{\Gamma \nvdash (A \vee B)_k, \Delta}\nvdash \vee r_2 \qquad \frac{\Gamma \nvdash (A)_k, (B)_k, \Delta}{\Gamma \nvdash (A \vee B)_k, \Delta}\nvdash \vee r_3$$

where $i < k$.

$$\frac{\Gamma, (A)_k \nvdash (B)_{>k}, \Delta}{\Gamma, (A \wedge B)_k \nvdash \Delta}\nvdash \wedge l_1 \qquad \frac{\Gamma, (B)_k \nvdash (A)_{>k}, \Delta}{\Gamma, (A \wedge B)_k \nvdash \Delta}\nvdash \wedge l_2$$

$$\frac{\Gamma, (A)_i \nvdash \Delta}{\Gamma \nvdash (A \wedge B)_k, \Delta}\nvdash \wedge r_1 \qquad \frac{\Gamma, (\neg A)_1 \nvdash \Delta}{\Gamma \nvdash (A \wedge B)_k, \Delta}\nvdash \wedge r_2 \qquad \frac{\Gamma, (B)_i \nvdash \Delta}{\Gamma \nvdash (A \wedge B)_k, \Delta}\nvdash \wedge r_3$$

$$\frac{\Gamma, (\neg B)_1 \nvdash \Delta}{\Gamma \nvdash (A \wedge B)_k, \Delta}\nvdash \wedge r_4 \qquad \frac{\Gamma \nvdash (A)_k, (B)_k, \Delta}{\Gamma \nvdash (A \wedge B)_k, \Delta}\nvdash \wedge r_5$$

where $i > k$.

The rules for ordered disjunction, with $k \leq opt_{\mathcal{L}}(A)$ and $l \leq opt_{\mathcal{L}}(B)$, are:

$$\frac{\Gamma, (A)_k \nvdash \Delta}{\Gamma, (A \vec{\times} B)_k \nvdash \Delta}\nvdash \vec{\times} l_1 \qquad \frac{\Gamma, (B)_l, (\neg A)_1 \nvdash \Delta}{\Gamma, (A \vec{\times} B)_{opt_{QCL}(A)+l} \nvdash \Delta}\nvdash \vec{\times} l_2$$

$$\frac{\Gamma \nvdash (A)_k, \Delta}{\Gamma \nvdash (A \vec{\times} B)_k, \Delta}\nvdash \vec{\times} r_1 \qquad \frac{\Gamma \nvdash (\neg A)_1, \Delta}{\Gamma \nvdash (A \vec{\times} B)_{opt_{\mathrm{QCL}}(A)+l}, \Delta}\nvdash \vec{\times} r_2 \qquad \frac{\Gamma \nvdash (B)_l, \Delta}{\Gamma \nvdash (A \vec{\times} B)_{opt_{\mathrm{QCL}}(A)+l}, \Delta}\nvdash \vec{\times} r_3$$

The degree overflow rules, with $k \in \mathbb{N}$, are:

$$\frac{\Gamma, \bot \nvdash \Delta}{\Gamma, (A)_{opt_{\mathrm{QCL}}(A)+k} \nvdash \Delta}\ {\nvdash}\, dol \qquad \frac{\Gamma \nvdash \Delta}{\Gamma \nvdash (A)_{opt_{\mathrm{QCL}}(A)+k}, \Delta}\ {\nvdash}\, dor$$

Example 7. The following is related to Example 4 and shows that the sequent $\neg(a \wedge b), ((a \overrightarrow{\times} b) \wedge (b \overrightarrow{\times} c))_2$ is satisfiable.

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots \\ (a \vee b), c, \neg b \nvdash a \wedge b, \bot\end{matrix}}{(a \vee b), (b \overrightarrow{\times} c)_2 \nvdash a \wedge b, \bot}\ {\nvdash}\, \overrightarrow{\times} l_2}{(b \overrightarrow{\times} c)_2 \nvdash \neg(a \overrightarrow{\times} b), a \wedge b, \bot}\ {\nvdash}\, \neg r}{((a \overrightarrow{\times} b) \wedge (b \overrightarrow{\times} c))_2 \nvdash a \wedge b, \bot}\ {\nvdash} \wedge l_2}{\neg(a \wedge b), ((a \overrightarrow{\times} b) \wedge (b \overrightarrow{\times} c))_2 \nvdash \bot}\ {\nvdash}\, \neg l$$

Note that the interpretation $\{a, c\}$ witnesses $(a \vee b), c, \neg b \nvdash a \wedge b, \bot$.

Proposition 3. $\mathbf{L}[\mathrm{QCL}]^-$ *is sound.*

Proof. The soundness of the negation rules is straightforward. The soundness of the rules $(\overrightarrow{\times} l_1)$, $(\overrightarrow{\times} l_2)$ and $(\overrightarrow{\times} r_1)$ follows by the same argument as for $\mathbf{L}[\mathrm{QCL}]$. For the remaining rules, it is easy to check that the same model witnessing the validity of the premise also witnesses the validity of the conclusion. □

Proposition 4. $\mathbf{L}[\mathrm{QCL}]^-$ *is complete.*

Proof. We show completeness by an induction over the (aggregated) formula complexity. Assume $\Gamma \nvdash \Delta$ is valid, i.e. $\Gamma \vdash \Delta$ is not valid. Now, there must be a rule in $\mathbf{L}[\mathrm{QCL}]$ for which $\Gamma \vdash \Delta$ is the conclusion. By the soundness of $\mathbf{L}[\mathrm{QCL}]$, this implies that at least one of the premises $\Gamma^* \vdash \Delta^*$ is not valid. However, then $\Gamma^* \nvdash \Delta^*$ is valid and, by induction, also provable. Now, by the construction of $\mathbf{L}[\mathrm{QCL}]^-$, there is a rule that allows us to derive $\Gamma \nvdash \Delta$ from $\Gamma^* \nvdash \Delta^*$. □

So far no cut-rule has been introduced for $\mathbf{L}[\mathrm{QCL}]^-$, and indeed, a counterpart of the cut rule would not be sound. One possibility is to introduce a contrapositive of cut as described by Bonatti and Olivetti [5]. Again, it is easy to see that this rule is admissible in our calculus:

$$\frac{\Gamma \nvdash \Delta \qquad \Gamma, (A)_k \vdash \Delta}{\Gamma \nvdash (A)_k, \Delta}\ cut2$$

We are now ready to combine $\mathbf{L}[\mathrm{QCL}]$ and $\mathbf{L}[\mathrm{QCL}]^-$ by defining an inference rule that allows us to go from labeled sequents to non-monotonic inferences. Again, we first consider the case where Γ is classical and A is a choice logic formula. The preferred model inference rule is:

$$\frac{\langle \Gamma, (A)_i \vdash \bot \rangle_{i<k} \qquad \Gamma, (A)_k \nvdash \bot \qquad \Gamma, (A)_k \vdash \Delta}{\Gamma, A \mathrel{|\!\!\sim}^{lex}_{\mathrm{QCL}} \Delta}\ {|\!\!\sim}_{simple}$$

Intuitively, the premises $\langle \Gamma, (A)_i \vdash \bot\rangle_{i<k}$ along with $\Gamma, (A)_k \nvdash \bot$ ensure that models satisfying A to a degree of k are preferred, while the premise $\Gamma, (A)_k \vdash \Delta$ ensures that Δ is entailed by those preferred models.

Example 8. The valid entailment $\neg(a \wedge b), (a \overrightarrow{\times} b) \wedge (b \overrightarrow{\times} c) \mathrel{|\!\!\sim}^{lex}_{\mathrm{QCL}} a \wedge c, b$ is provable by choosing $k = 2$:

$$\dfrac{\overset{(\varphi_1)}{\Gamma, ((a \overrightarrow{\times} b) \wedge (b \overrightarrow{\times} c))_1 \vdash \bot} \qquad \overset{(\varphi_2)}{\Gamma, ((a \overrightarrow{\times} b) \wedge (b \overrightarrow{\times} c))_2 \nvdash \bot} \qquad \overset{(\varphi_3)}{\Gamma, ((a \overrightarrow{\times} b) \wedge (b \overrightarrow{\times} c))_2 \vdash \Delta}}{\Gamma, (a \overrightarrow{\times} b) \wedge (b \overrightarrow{\times} c) \mathrel{|\!\!\sim}^{lex}_{\mathrm{QCL}} \Delta} \; {\mathrel{|\!\!\sim}_{simple}}$$

with $\Gamma = \neg(a \wedge b)$ and $\Delta = a \wedge c, b$. φ_3 is the $\mathbf{L}$[QCL]-proof from Example 4 and φ_2 is the $\mathbf{L}[\mathrm{QCL}]^-$-proof from Example 7. φ_1 is not shown explicitly, but it can be verified that the corresponding sequent is provable.

We extend $\mathrel{|\!\!\sim}_{simple}$ to the more general case, where more than one non-classical formula may be present, to obtain a calculus for preferred model entailment. An additional rule $\mathrel{|\!\!\sim}_{unsat}$ is needed in case a theory is classically unsatisfiable.

Definition 14 ($\mathbf{L}[\mathrm{QCL}]^{lex}_{|\!\sim}$). *Let $\leq_l$ be the order on vectors in $\mathbb{N}^k$ defined by*

- $\boldsymbol{v} <_l \boldsymbol{w}$ *if there is some $n \in \mathbb{N}$ such that $\boldsymbol{v}$ has more entries of value n and for all $1 \leq m < n$ both vectors have the same number of entries of value m.*
- $\boldsymbol{v} =_l \boldsymbol{w}$ *if, for all $n \in \mathbb{N}$, $\boldsymbol{v}$ and $\boldsymbol{w}$ have the same number of entries of value n.*

$\mathbf{L}[\mathrm{QCL}]^{lex}_{|\!\sim}$ *consists of the axioms and rules of* $\mathbf{L}$[QCL] *and* $\mathbf{L}[\mathrm{QCL}]^-$ *plus the following rules, where $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{N}^k$, Γ consists of only classical formulas, and every A_i with $1 \leq i \leq k$ is a* QCL*-formula:*

$$\dfrac{\langle \Gamma, (A_1)_{w_1}, \dots, (A_k)_{w_k} \vdash \bot\rangle_{\boldsymbol{w} < \boldsymbol{v}} \qquad \Gamma, (A_1)_{v_1}, \dots, (A_k)_{v_k} \nvdash \bot \qquad \langle \Gamma, (A_1)_{w_1}, \dots, (A_k)_{w_k} \vdash \Delta\rangle_{\boldsymbol{w} = \boldsymbol{v}}}{\Gamma, A_1, \dots, A_k \mathrel{|\!\!\sim}^{lex}_{\mathrm{QCL}} \Delta} \; {\mathrel{|\!\!\sim}_{lex}}$$

$$\dfrac{\Gamma, cp(A_1), \dots, cp(A_k) \vdash \bot}{\Gamma, A_1, \dots, A_k \mathrel{|\!\!\sim}^{lex}_{\mathrm{QCL}} \Delta} \; {\mathrel{|\!\!\sim}_{unsat}}$$

We first provide a small example and then show soundness and completeness.

Example 9. Consider the valid entailment $\neg(a \wedge b), (a \overrightarrow{\times} b), (b \overrightarrow{\times} c) \mathrel{|\!\!\sim}^{lex}_{\mathrm{QCL}} a \wedge c, b$ similar to Example 8, but with the information that we require $(a \overrightarrow{\times} b)$ and $(b \overrightarrow{\times} c)$ encoded as separate formulas. It is not possible to satisfy all formulas on the left to a degree of 1. Rather, it is optimal to either satisfy $(\neg(a \wedge b))_1, (a \overrightarrow{\times} b)_1, (b \overrightarrow{\times} c)_2$ or, alternatively, $(\neg(a \wedge b))_1, (a \overrightarrow{\times} b)_2, (b \overrightarrow{\times} c)_1$. We choose $\boldsymbol{v} = (1, 1, 2)$, with $\boldsymbol{w} = (1, 1, 1)$ being the only vector $\boldsymbol{w}$ s.t. $\boldsymbol{w} < \boldsymbol{v}$. Thus, we get

$$\dfrac{\overset{\vdots}{\Gamma, (a \overrightarrow{\times} b)_1, (b \overrightarrow{\times} c)_1 \vdash \bot} \qquad \overset{\vdots}{\Gamma, (a \overrightarrow{\times} b)_1, (b \overrightarrow{\times} c)_2 \nvdash \bot} \qquad \overset{\vdots}{\Gamma, (a \overrightarrow{\times} b)_1, (b \overrightarrow{\times} c)_2 \vdash \Delta} \qquad \overset{\vdots}{\Gamma, (a \overrightarrow{\times} b)_2, (b \overrightarrow{\times} c)_1 \vdash \Delta}}{\Gamma, (a \overrightarrow{\times} b), (b \overrightarrow{\times} c) \mathrel{|\!\!\sim}^{lex}_{\mathrm{QCL}} \Delta} \; {\mathrel{|\!\!\sim}_{lex}}$$

with $\Gamma = \neg(a \wedge b)$ and $\Delta = a \wedge c, b$. It can be verified that indeed all branches are provable, but we do not show this explicitly here.

Proposition 5. $\mathbf{L}[\mathrm{QCL}]^{lex}_{\mid\sim}$ *is sound.*

Proof. Consider first the $\mid\sim_{lex}$-rule and assume that all premises are derivable. By the soundness of $\mathbf{L}[\mathrm{QCL}]$ and $\mathbf{L}[\mathrm{QCL}]^-$ they are also valid. From the first set of premises $\langle \Gamma, (A_1)_{w_1}, \ldots, (A_k)_{w_k} \vdash \bot \rangle_{\boldsymbol{w}<\boldsymbol{v}}$ we can conclude that if there is some model M of Γ that satisfies A_i to a degree of v_i for all $1 \leq i \leq k$, then $M \in \mathit{Prf}^{lex}_{\mathrm{QCL}}(\Gamma \cup \{A_1, \ldots, A_k\})$. The premise $\Gamma, (A_1)_{v_1}, \ldots, (A_k)_{v_k} \nvdash \bot$ ensures that there is such a model M. By the last set of premises $\langle \Gamma, (A_1)_{w_1}, \ldots, (A_k)_{w_k} \vdash \Delta \rangle_{\boldsymbol{w}=\boldsymbol{v}}$, we can conclude that all models of $\Gamma \cup \{A_1, \ldots, A_k\}$ that are equally as preferred as M, i.e., all $M' \in \mathit{Prf}^{lex}_{\mathrm{QCL}}(\Gamma \cup \{A_1, \ldots, A_k\})$, satisfy at least one formula in Δ. Therefore, $\Gamma, A_1, \ldots, A_k \mid\sim^{lex}_{\mathrm{QCL}} \Delta$ is valid.

Now consider the $\mid\sim_{unsat}$-rule and assume that $\Gamma, \mathit{cp}(A_1), \ldots, \mathit{cp}(A_k) \vdash \bot$ is derivable and therefore valid. Thus, $\Gamma \cup \{A_1, \ldots, A_k\}$ has no models and therefore also no preferred models. Then $\Gamma, A_1, \ldots, A_k \mid\sim^{lex}_{\mathrm{QCL}} \Delta$ is valid. □

Proposition 6. $\mathbf{L}[\mathrm{QCL}]^{lex}_{\mid\sim}$ *is complete.*

Proof. Assume that $\Gamma, A_1, \ldots, A_k \mid\sim^{lex}_{\mathrm{QCL}} \Delta$ is valid. If $\Gamma \cup \{A_1, \ldots, A_k\}$ is unsatisfiable then $\Gamma, \mathit{cp}(A_1), \ldots, \mathit{cp}(A_k) \vdash \bot$ is valid, i.e., we can apply the $\mid\sim_{unsat}$-rule. Now consider the case that $\Gamma \cup \{A_1, \ldots, A_k\}$ is satisfiable and assume that some preferred model M of $\Gamma \cup \{A_1, \ldots, A_k\}$ satisfies A_i to a degree of v_i for all $1 \leq i \leq k$. Then, we claim that all premises of the rule are valid and, by the completeness of $\mathbf{L}[\mathrm{QCL}]$ and $\mathbf{L}[\mathrm{QCL}]^-$, also derivable.

Assume by contradiction that one of the premises is not valid. First, consider the case that $\Gamma, (A_1)_{w_1}, \ldots, (A_k)_{w_k} \vdash \bot$ is not valid for some $\boldsymbol{w} < \boldsymbol{w}$. Then there is a model M' of Γ that satisfies A_i to a degree of w_i for all $1 \leq i \leq k$. However, this contradicts the assumption that M is a preferred model of $\Gamma \cup \{A_1, \ldots, A_k\}$.

Next, assume that $\Gamma, (A_1)_{v_1}, \ldots, (A_k)_{v_k} \nvdash \bot$ is not valid. However, M satisfies $\Gamma, (A_1)_{v_1}, \ldots, (A_k)_{v_k}$ and does not satisfy $\bot$. Contradiction.

Finally, we assume that $\Gamma, (A_1)_{w_1}, \ldots, (A_k)_{w_k} \vdash \Delta$ is not valid for some $\boldsymbol{w} = \boldsymbol{v}$. Then, there is a model M' of Γ that satisfies A_i to a degree of w_i for all $1 \leq i \leq k$ but does not satisfy any formula in Δ. But M' is a preferred model of $\Gamma \cup \{A_1, \ldots, A_k\}$, which contradicts $\Gamma, A_1, \ldots, A_k \mid\sim^{lex}_{\mathrm{QCL}} \Delta$ being valid. □

In this paper, we focused on the lexicographic semantics for preferred models of choice logic theories. However, rules for other semantics, e.g. a rule $\mid\sim_{inc}$ for the inclusion based approach (cf. Definition 8), can be obtained by simply adapting the way in which vectors over $\mathbb{N}^k$ are compared (cf. Definition 14).

5 Beyond QCL

QCL was the first choice logic to be described [7], and applications concerned with QCL and ordered disjunction have been discussed in the literature [3,8,13]. For this reason, the main focus in this paper lies with QCL. However, as we have seen in Sect. 2, CCL and its ordered conjunction show that interesting logics similar to QCL exist. We will now demonstrate that $\mathbf{L}[\mathrm{QCL}]$ can easily be adapted for other choice logics. In particular, we introduce **L**[CCL] in which the rules of **L**[QCL] for the classical connectives can be retained. All that is needed is to replace the $\vec{\times}$-rules by appropriate rules for the choice connective $\vec{\odot}$ of CCL.

Definition 15 (L[CCL]). **L**[CCL] *is* **L**[QCL]*, except that the* $\vec{\times}$*-rules are replaced by the following* $\vec{\odot}$*-rules:*

$$\frac{\Gamma, (A)_1, (B)_k \vdash \Delta}{\Gamma, (A\vec{\odot}B)_k \vdash \Delta}\,\vec{\odot}l_1 \qquad \frac{\Gamma, (A)_l, (\neg B)_1 \vdash \Delta}{\Gamma, (A\vec{\odot}B)_{opt_{\mathrm{CCL}}(B)+l} \vdash \Delta}\,\vec{\odot}l_2 \qquad \frac{\Gamma, (A)_m \vdash \Delta}{\Gamma, (A\vec{\odot}B)_{opt_{\mathrm{CCL}}(B)+m} \vdash \Delta}\,\vec{\odot}l_3$$

$$\frac{\Gamma \vdash (A)_1, \Delta \qquad \Gamma \vdash (B)_k, \Delta}{\Gamma \vdash (A\vec{\odot}B)_k, \Delta}\,\vec{\odot}r_1 \quad \frac{\Gamma \vdash (A)_l, \Delta \qquad \Gamma \vdash (\neg B)_1, \Delta}{\Gamma \vdash (A\vec{\odot}B)_{opt_{\mathrm{CCL}}(B)+l}, \Delta}\,\vec{\odot}r_2 \quad \frac{\Gamma \vdash (A)_m, \Delta}{\Gamma \vdash (A\vec{\odot}B)_{opt_{\mathrm{CCL}}(B)+m}, \Delta}\,\vec{\odot}r_3$$

where $k \leq opt_{\mathrm{CCL}}(B)$*,* $l \leq opt_{\mathrm{CCL}}(A)$*, and* $1 < m \leq opt_{\mathrm{CCL}}(A)$*.*

Note that, given $\Gamma, (A\vec{\odot}B)_{opt_{\mathrm{CCL}}(B)+m} \vdash \Delta$ with $1 < m \leq opt_{\mathrm{CCL}}(A)$, we need to guess whether $\vec{\odot}l_2$ or $\vec{\odot}l_3$ has to be applied. We do not define **L**[CCL]$^-$ here, but the necessary rules for $\vec{\odot}$ can be inferred from the $\vec{\odot}$-rules of **L**[CCL] in a similar way to how **L**[QCL]$^-$ was derived from **L**[QCL].

Proposition 7. **L**[CCL] *is sound.*

Proof. We consider the newly introduced rules.

- For $\vec{\odot}l_1$, $\vec{\odot}l_2$, and $\vec{\odot}l_3$ this follows directly from the definition of CCL.
- ($\vec{\odot}r_1$). Assume both premises are valid, i.e., every model of Γ is a model of Δ or of $(A)_1$ and $(B)_k$ with $k \leq opt_{\mathcal{L}}(B)$. By definition, any model that satisfies $(A)_1$ and $(B)_k$ satisfies $A\vec{\odot}B$ to degree k. Thus, every model of Γ is a model of Δ or of $(A\vec{\odot}B)_k$, which means the conclusion of the rule is valid.
- ($\vec{\odot}r_2$). Assume both premises are valid, i.e., every model of Γ is either a model of Δ or of $(A)_l$ and $(\neg B)_1$ with $l \leq opt_{\mathrm{CCL}}(A)$. By definition, any model that satisfies $(A)_l$ and does not satisfy B (and hence satisfies $(\neg B)_1$) satisfies $A\vec{\odot}B$ to degree $opt_{\mathrm{CCL}}(B) + l$.
- ($\vec{\odot}r_3$). Assume that the premise is valid, i.e., every model of Γ is either a model of Δ or of $(A)_m$ with $1 < m \leq opt_{\mathrm{CCL}}(A)$. By definition, any model that satisfies $(A)_m$, regardless of what degree this model ascribes to B, satisfies $A\vec{\odot}B$ to degree $opt_{\mathrm{CCL}}(B) + m$. □

Proposition 8. **L**[CCL] *is complete.*

Proof. We adapt the induction of the proof of Proposition 2:

- Assume that a sequent of the form $\Gamma, (A\vec{\odot}B)_k \vdash \Delta$ is valid, with $k \leq opt_{\mathcal{L}}(B)$. All models that satisfy $(A\vec{\odot}B)_k$ must satisfy A to a degree of 1 and B to a degree of k. Thus, $\Gamma, (A)_1, (B)_k \vdash \Delta$ is valid, and, by the induction hypothesis, $\Gamma, (A\vec{\odot}B)_k \vdash \Delta$ is provable. Similarly for the cases $\Gamma, (A\vec{\odot}B)_{opt_{\mathrm{CCL}}(B)+l} \vdash \Delta$ with $l \leq opt_{\mathrm{CCL}}(A)$, and $\Gamma, (A\vec{\odot}B)_{opt_{\mathrm{CCL}}(B)+m} \vdash \Delta$ with $1 < m \leq opt_{\mathrm{CCL}}(A)$.

- Assume that a sequent of the form $\Gamma \vdash (A \overrightarrow{\odot} B)_k, \Delta$ is valid, with $k \leq opt_{\mathcal{L}}(B)$. We claim that then $\Gamma \vdash (A)_1, \Delta$ and $\Gamma \vdash (B)_k, \Delta$ are valid. Assume, for the sake of a contradiction, that the first sequent is not valid. This means that there is a model M of Γ that is neither a model of $(A)_1$ nor of Δ. However, then M satisfies $A \overrightarrow{\odot} B$ to a degree higher than $opt_{\mathrm{CCL}}(B)$. This contradicts the assumption that $\Gamma \vdash (A \overrightarrow{\odot} B)_k, \Delta$ is valid. Assume now that the second sequent is not valid, i.e., that there is a model M of Γ that is neither a model of $(B)_k$ nor of Δ. Then M cannot be a model of $(A \overrightarrow{\odot} B)_k$, contradicting the assumption. As before, it follows by the induction hypothesis that $\Gamma \vdash (A \overrightarrow{\odot} B)_k, \Delta$ is provable. Similarly for the cases $\Gamma \vdash (A \overrightarrow{\odot} B)_{opt_{\mathrm{CCL}}(B)+l}, \Delta$ with $l \leq opt_{\mathrm{CCL}}(A)$, and $\Gamma \vdash (A \overrightarrow{\odot} B)_{opt_{\mathrm{CCL}}(B)+m}, \Delta$ with $1 < m \leq opt_{\mathrm{CCL}}(A)$. □

We are confident that our methods can be adapted not only for QCL and CCL, but for numerous other instantiations of the choice logic framework defined in Sect. 2. We mention here *lexicographic choice logic* (LCL) [4], in which $A \overrightarrow{\diamond} B$ expresses that it is best to satisfy A and B, second best to satisfy only A, third best to satisfy only B, and unacceptable to satisfy neither.

Moreover, note that the inference rules $\mid\!\sim_{lex}$ and $\mid\!\sim_{unsat}$ (cf. Definition 14) do not depend on any specific choice logic. Thus, once labeled calculi are developed for a choice logic, a calculus for preferred model entailment follows immediately.

6 Conclusion

In this paper we introduce a sound and complete sequent calculus for preferred model entailment in QCL. This non-monotonic calculus is built on two calculi: a monotonic labeled sequent calculus and a corresponding refutation calculus.

Our systems are modular and can easily be adapted: on the one hand, calculi for choice logics other than QCL can be obtained by introducing suitable rules for the choice connectives of the new logic, as exemplified with our calculus for CCL; on the other hand, a non-monotonic calculus for preferred model semantics other than the lexicographic semantics can be obtained by adapting the inference rule $\mid\!\sim_{lex}$ which transitions from preferred model entailment to the labeled calculi.

Our work contributes to the line of research on non-monotonic sequent calculi that make use of refutation systems [5]. Our system is the first proof calculus for choice logics, which have been studied mainly from the viewpoint of their computational properties [4] and their potential applications [3,8,13] so far.

Regarding future work, we aim to investigate the proof complexity of our calculi, and how this complexity might depend on which choice logic or preferred model semantics is considered. Also, calculi for other choice logics such as LCL could be explicitly defined, as was done with CCL in Sect. 5.

Acknowledgments. We thank the anonymous reviewers for their valuable feedback. This work was funded by the Austrian Science Fund (FWF) under the grants Y698 and J4581.

References

1. Avron, A.: The method of hypersequents in the proof theory of propositional non-classical logics. In: Hodges, W., Hyland, M., Steinhorn, C., Truss, J. (eds.) Logic: From Foundations to Applications, pp. 1–32. Oxford Science Publications, Oxford (1996)
2. Baaz, M., Lahav, O., Zamansky, A.: Finite-valued semantics for canonical labelled calculi. J. Autom. Reason. **51**(4), 401–430 (2013)
3. Benferhat, S., Sedki, K.: Alert correlation based on a logical handling of administrator preferences and knowledge. In: SECRYPT, pp. 50–56. INSTICC Press (2008)
4. Bernreiter, M., Maly, J., Woltran, S.: Choice logics and their computational properties. In: IJCAI, pp. 1794–1800. ijcai.org (2021)
5. Bonatti, P.A., Olivetti, N.: Sequent calculi for propositional nonmonotonic logics. ACM Trans. Comput. Log. **3**(2), 226–278 (2002)
6. Boudjelida, A., Benferhat, S.: Conjunctive choice logic. In: ISAIM (2016)
7. Brewka, G., Benferhat, S., Berre, D.L.: Qualitative choice logic. Artif. Intell. **157**(1–2), 203–237 (2004)
8. Brewka, G., Niemelä, I., Syrjänen, T.: Logic programs with ordered disjunction. Comput. Intell. **20**(2), 335–357 (2004)
9. Carnielli, W.A.: Systematization of finite many-valued logics through the method of tableaux. J. Symb. Log. **52**(2), 473–493 (1987)
10. Geibinger, T., Tompits, H.: Sequent-type calculi for systems of nonmonotonic paraconsistent logics. In: ICLP Technical Communications. EPTCS, vol. 325, pp. 178–191 (2020)
11. Governatori, G., Rotolo, A.: Logic of violations: a Gentzen system for reasoning with contrary-to-duty obligations. Australas. J. Logic **4**, 193–215 (2006)
12. Kaminski, M., Francez, N.: Calculi for many-valued logics. Log. Univers. **15**(2), 193–226 (2021)
13. Liétard, L., Hadjali, A., Rocacher, D.: Towards a gradual QCL model for database querying. In: Laurent, A., Strauss, O., Bouchon-Meunier, B., Yager, R.R. (eds.) IPMU 2014. CCIS, vol. 444, pp. 130–139. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08852-5_14
14. McCarthy, J.: Circumscription - a form of non-monotonic reasoning. Artif. Intell. **13**(1–2), 27–39 (1980)
15. Moore, R.C.: Semantical considerations on nonmonotonic logic. Artif. Intell. **25**(1), 75–94 (1985)
16. Reiter, R.: A logic for default reasoning. Artif. Intell. **13**(1–2), 81–132 (1980)
17. Tiomkin, M.L.: Proving unprovability. In: LICS, pp. 22–26. IEEE Computer Society (1988)

Lash 1.0 (System Description)

Chad E. Brown[1] and Cezary Kaliszyk[2](✉)

[1] Czech Technical University in Prague, Prague, Czech Republic
[2] University of Innsbruck, Innsbruck, Austria
cezary.kaliszyk@uibk.ac.at

Abstract. Lash is a higher-order automated theorem prover created as a fork of the theorem prover Satallax. The basic underlying calculus of Satallax is a ground tableau calculus whose rules only use shallow information about the terms and formulas taking part in the rule. Lash uses new, efficient C representations of vital structures and operations. Most importantly, Lash uses a C representation of (normal) terms with perfect sharing along with a C implementation of normalizing substitutions. We describe the ways in which Lash differs from Satallax and the performance improvement of Lash over Satallax when used with analogous flag settings. With a 10 s timeout Lash outperforms Satallax on a collection TH0 problems from the TPTP. We conclude with ideas for continuing the development of Lash.

Keywords: Higher-order logic · Automated reasoning · TPTP

1 Introduction

Satallax [4,7] is an automated theorem prover for higher-order logic that was a top competitor in the THF division of CASC [10] for most of the 2010s. The basic calculus of Satallax is a complete ground tableau calculus [2,5,6]. In recent years the top systems of the THF division of CASC are primarily based on resolution and superposition [3,8,11]. At the moment it is an open question whether there is a research and development path via which a tableau based prover could again become competitive. As a first step towards answering this question we have created a fork of Satallax, called Lash, focused on giving efficient C implementations of data structures and operations needed for search in the basic calculus.

Satallax was partly competitive due to (optional) additions that went beyond the basic calculus. Three of the most successful additions were the use of higher-order pattern clauses during search, the use of higher-order unification as a heuristic to suggest instantiations at function types and the use of the first-order theorem prover E as a backend to try to prove the first-order part of the current state is already unsatisfiable. Satallax includes flags that can be used to activate or deactivate such additions so that search only uses the basic calculus. They are deactivated by default. Satallax has three representations of terms in Ocaml. The basic calculus rules use the primary representation. Higher-order

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 350–358, 2022.
https://doi.org/10.1007/978-3-031-10769-6_21

unification and pattern clauses make use of a representation that includes a case for metavariables to be instantiated. Communication with E uses a third representation restricted to first-order terms and formulas. When only the basic calculus is used, only the primary representation is needed.

Assuming only the basic calculus is used only limited information about (normal) terms is needed during the search. Typically we only need to know the outer structure of the principal formulas of each rule, and so the full term does not need to be traversed. In some cases Satallax either implicitly or explicitly traverses the term. The implicit cases are when a rule needs to know if two terms are equal. In Satallax, Ocaml's equality is used to test for equality of terms, implicitly relying on a recursion over the term. The explicit cases are quantifier rules that instantiate with either a term or a fresh constant. In the former case we may also need to normalize the result after instantiating with a term.

In order to give an optimized implementation of the basic calculus we have created a new theorem prover, Lash[1], by forking a recent version of Satallax (Satallax 3.4), the last version that won the THF division of CASC (in 2019). Generally speaking, we have removed all the additional code that goes beyond the basic calculus. In particular we do not need terms with metavariables since we support neither pattern clauses nor higher-order unification in Lash. Likewise we do not need a special representation for first-order terms and formulas since Lash does not communicate with E. We have added efficient C implementations of (normal) terms with perfect sharing. Additionally we have added new efficient C implementations of priority queues and the association of formulas with integers (to communicate with MiniSat). To measure the speedup given by the new parts of the implementation we have run Satallax 3.4 using flag settings that only use the basic calculus and Lash 1.0 using the same flag settings. We have also compared Lash to Satallax 3.4 using Satallax's default strategy with a timeout of 10 s, and have found that Lash 1.0 outperforms Satallax with this short timeout even when Satallax is using the optional additions (including calling E). We describe the changes and present a number of examples for which the changes lead to a significant speedup.

2 Preliminaries

We will presume a familiarity with simple type theory and only give a quick description to make our use of notation clear, largely following [6]. We assume a set of base types, one of which is the type o of propositions (also called booleans), and the rest we refer to as sorts. We use α, β to range over sorts and σ, τ to range over types. The only types other than base types are function types $\sigma\tau$, which can be thought of as the type of functions from σ to τ.

All terms have a unique type and are inductively defined as (typed) variables, (typed) constants, well-typed applications $(t\ s)$ and λ-abstractions $(\lambda x.t)$. We

[1] Lash 1.0 along with accompanying material is available at http://grid01.ciirc.cvut.cz/~chad/ijcar2022lash/.

also include the logical constant $\bot$ as a term of type o, terms (of type o) of the form $(s \Rightarrow t)$ (implications) and $(\forall x.t)$ (universal quantifiers) where s, t have type o and terms (of type o) of the form $(s =_\sigma t)$ where s, t have a common type σ. We also include choice constants ε_σ of $(\sigma o)\sigma$ at each type σ. We write $\neg t$ for $t \Rightarrow \bot$ and $(s \neq_\sigma t)$ for $(s =_\sigma t \Rightarrow \bot)$. We omit type parentheses and type annotations except where they are needed for clarity. Terms of type o are also called propositions. We also use $\top$, $\vee$, $\wedge$, $\exists$ with the understanding that these are notations for equivalent propositions in the set of terms above.

We assume terms are equal if they are the same up to α-conversion of bound variables (using de Bruijn indices in the implementation). We write $[s]$ for the $\beta\eta$-normal form of s.

The tableau calculi of [6] (without choice) and [2] (with choice) define when a branch is refutable. A branch is a finite set of normal propositions. We let A range over branches and write A, s for the branch $A \cup \{s\}$. We will not give a full calculus, but will instead discuss a few of the rules with surprising properties. Before doing so we emphasize rules that are *not* in the calculus. There is no cut rule stating that if A, s and $A, \neg s$ are refutable, then A is refutable. (During search such a rule would require synthesizing the cut formula s.) There is also no rule stating that if the branch $A, (s = t), [ps], [pt]$ is refutable, then $A, (s = t), [ps]$ is refutable (where s, t have type σ and p is a term of type σo). That is, there is no rule for rewriting into arbitrarily deep positions using equations.

All the tableau rules only need to examine the outer structure to test if they apply (when searching backwards for a refutation). When applying the rule, new formulas are constructed and added to the branch (or potentially multiple branches, each a subgoal to be refuted). An example is the confrontation rule, the only rule involving positive equations. The confrontation rule states that if $s =_\alpha t$ and $u \neq_\alpha v$ are on a branch A (where α is a sort), then we can refute A by refuting $A, s \neq u, t \neq u$ and $A, s \neq v, t \neq v$. A similar rule is the mating rule, which states that if $ps_1 \ldots s_n$ and $\neg pt_1 \ldots t_n$ are on a branch A (where p is a constant of type $\sigma_1 \cdots \sigma_n o$), then we can refute A by refuting each of the branches $A, s_i \neq t_i$ for each $i \in \{1, \ldots, n\}$. The mating rule demonstrates how disequations can appear on a branch even if the original branch to refute contained no reference to equality at all. One way a branch can be closed is if $s \neq s$ is on the branch. In an implementation, this means an equality check is done for s and t whenever a disequation $s \neq t$ is added to the branch. In Satallax this requires Ocaml to traverse the terms. In Lash this only requires comparing the unique integer ids the implementation assigns to the terms.

The disequations generated on a branch play an important role. Terms (of sort α) occuring on one side of a disequation on a branch are called *discriminating terms*. The rule for instantiating a quantified formula $\forall x.t$ (where x has sort α) is restricted to instantiating with discriminating terms (or a default term if no terms of sort α are discriminating). During search in Satallax this means there is a finite set of permitted instantiations (at sort α) and this set grows as disequations are produced. Note that, unlike most automated theorem provers, the instantiations do not arise from unification. In Satallax (and Lash) when

$\forall x.t$ is being processed it is instantiated with all previously processed instantiations. When a new instantiation is produced, previously processed universally quantified propositions are instantiated with it. When $\forall x.t$ is instantiated with s, then $[(\lambda x.t)s]$ is added to the branch. Such an instantiation is the important case where the new formula involves term traversals: both for substitution and normalization. In Satallax the substitution and normalization require multiple term traversals. In Lash we have used normalizing substitutions and memorized previous computations, minimizing the number of term traversals. The need to instantiate arises when processing either a universally quantified proposition (giving a new quantifier to instantiate) or a disequation at a sort (giving new discriminating terms).

We discuss a small example both Satallax and Lash can easily prove. We briefly describe what both do in order to give the flavor of the procedure and (hopefully) prevent readers from assuming the provers behave too similarly from readers based on other calculi (e.g., resolution).

Example `SEV241^5` from TPTP v7.5.0 [9] (`X5201A` from Tps [1]) contains a minor amount of features going beyond first-order logic. The statement to prove is

$$\forall x. U\ x \wedge W\ x \Rightarrow \forall S. (S = U \vee S = W) \Rightarrow Sx.$$

Here U and W are constants of type αo, x is a variable of type α and S is a variable of type αo. The higher-order aspects of this problem are the quantifier for S (though this could be circumvented by making S a constant like U and W) and the equations between predicates (though these could be circumvented by replacing $S = U$ by $\forall y. Sy \Leftrightarrow Uy$ and replacing $S = W$ similarly). The tableau rules effectively do both during search.

Satallax never clausifies. The formula above is negated and assumed. We will informally describe tableau rules as splitting the problem into subgoals, though this is technically mediated through MiniSat (where the set of MiniSat clauses is unsatisfiable when all branches are closed). Tableau rules are applied until the problem involves a constant c (for x), a constant S' for S and assumptions $U\ c$, $W\ c$, $S' = U \vee S' = W$ and $\neg S'c$ on the branch. The disjunction is internally $S' \neq U \Rightarrow S' = W$ and the implication rule splits the problem into two branches, one with $S' = U$ and one with $S' = W$. Both branches are solved in analogous ways and we only describe the $S' = U$ branch. Since $S' = U$ is an equation at function type, the relevant rule adds $\forall y. S'y = Uy$ to the branch. Since there are no disequations on the branch, there is no instantiation available for $\forall y. S'y = Uy$. In such a case, a default instantiation is created and used. That is, a default constant d (of sort α) is generated and we instantiate with this d, giving $S'd =_o Ud$. The rule for equations at type o splits into two subgoals: one branch with $S'd$ and Ud and another with $\neg S'd$ and $\neg Ud$. On the first branch we mate $S'd$ with $\neg S'c$ adding the disequation $d \neq c$ to the branch. This makes c available as an instantiation for $\forall y. S'y = Uy$. After instantiating with c the rest of the subcase is straightforward. In the other subgoal we mate $U\ c$ with $\neg Ud$ giving the disequation $c \neq d$. Again, c becomes available as an instantiation and the rest of the subcase is straightforward.

3 Terms with Perfect Sharing

Lash represents normal terms as C structures, with a unique integer id assigned to each term. The structure contains a tag indicating which kind of term is represented, a number that is used to either indicate the de Bruijn index (for a variable), the name (for a constant), or the type (for a λ-abstraction, a universal quantifier, a choice operator, or an equation). Two pointers (optionally) point to relevant subterms in each case. In addition the structure maintains the information of which de Bruijn indices are free in the term (with de Bruijn indices limited to a maximum of 255). Knowing the free de Bruijn indices of terms makes recognizing potential η-redexes possible without traversing the λ-abstraction. Likewise it is possible to determine when shifting and substitution of de Bruijn indices would not affect a term, avoiding the need to traverse the term.

In Ocaml only the unique integer id is directly revealed and this is sufficient to test for equality of terms. Hash tables are used to uniquely assign types to integers and strings (for names) to integers and these integers are used to interface with the C code. Various functions are used in the Ocaml-C interface to request the construction of (normal) terms. For example, given the two Ocaml integer ids i and j corresponding to terms s and t, the function `mk_norm_ap` given i and j will return an integer k corresponding to the normal term $[s\ t]$. The C implementation recognizes if s is a λ-abstraction and performs all $\beta\eta$-reductions to obtain a normal term. Additionally, the C implementation treats terms as graphs with perfect sharing, and additionally caches previous operations (including substitutions and de Bruijn shifting) to prevent recomputation.

In addition to the low-level C term reimplementation, we have also provided a number of other low-level functionalities replacing the slower parts of the Ocaml code. This includes low-level priority queues, as well as C code used to associate the integers representing normal propositions with integers that are used to communicate with MiniSat. The MiniSat integers are nonzero and satisfy the property that minus on integers corresponds to negation of propositions.

4 Results and Examples

The first mode in the default schedule for Satallax 3.1 is MODE213. This mode activates one feature that goes beyond the basic calculus: pattern clauses. Additionally the mode sets a flag that tries to split the initial goal into several independent subgoals before beginning the search proper. Through experimentation we have found that setting a flag (common to both Satallax and Lash) to essentially prevent MiniSat from searching (i.e., only using MiniSat to recognize contradictions that are evident without search) often improves the performance. We have created a modified mode MODE213D that deactivates these additions (and delays the use of MiniSat) so that Satallax and Lash will have a similar (and often the same) search space. (Sometimes the search spaces differ due to differences in the way Satallax and Lash enumerate instantiations for function types, an issue we

Table 1. Lash vs. Satallax on 2053 TH0 Problems.

Prover	Problems Solved
Lash	1501 (73%)
Satallax (with E)	1487 (72%)
Satallax (without E)	1445 (70%)
Satallax (Lash Schedule)	1412 (69%)

will not focus on here.) We have also run Lash with many variants of Satallax modes with similar modifications. From such test runs we have created a 10 s schedule consisting of 5 modes.

To give a general comparison of Satallax and Lash we have run both on 2053 TH0 problems from a recent release of the TPTP [9] (7.5.0). We initially selected all problems with TPTP status of Theorem or Unsatisfiable (so they should be provable in principle) without polymorphism (or similar extensions of TH0). We additionally removed a few problems that could not be parsed by Satallax 3.4 and removed a few hundred problems big enough to activate SINE in Satallax 3.4.

We ran Lash for 10 s with its default schedule over this problem set. For comparison, we have run Satallax 3.4 for 10 s in three different ways: using the Lash schedule (since the flag settings make sense for both systems) and using Satallax 3.4's default schedule both with and without access to E [12]. The results are reported in Table 1. It is already promising that Lash has the ability to slightly outperform Satallax even when Satallax is allowed to call E.

To get a clearer view of the improvement we discuss a few specific examples.

TPTP problem `NUM638^1` (part of Theorem 3 from the AUTOMATH formalization of Landau's book) is about the natural numbers (starting from 1). The problem assumes a successor function s is injective and that every number other than 1 has a predecessor. An abstract notion of existence is used by having a constant some of type $(\iota o)o$ about which no extra assumptions are made, so the assumption is formally $\forall x.x \neq 1 \Rightarrow \mathsf{some}(\lambda u.x = su)$. For a fixed n, $n \neq 1$ is assumed and the conjecture to prove is the negation of the implication $(\forall xy.n = sx \Rightarrow n = sy \Rightarrow x = y) \Rightarrow \neg(\mathsf{some}(\lambda u.n = su))$. The implication is assumed and the search must rule out the negation of the antecedent (i.e., that n has two predecessors) and the succedent (that n has no predecessor). Satallax and Lash both take 3911 steps to prove this example. With MODE213D, Lash completes the search in 0.4 s while Satallax requires almost 29 s.

TPTP problem `SEV108^5` (`SIX_THEOREM` from TPS [1]) corresponds to proving the Ramsey number R(3,3) is at most 6. The problem assumes there is a symmetric binary relation R (the edge relation of a graph with the sort as vertices) and there are (at least) 6 distinct elements. The conclusion is that there are either 3 distinct elements all of which are R-related or 3 distinct elements none of which are R-related. Satallax and Lash can solve the problem in 14129 steps with mode MODE213D. Satallax proves the theorem in 0.153 s while Lash proves the theorem in the same number of steps but in 0.046 s.

The difference is more impressive if we consider the modified problem of proving R(3, 4) is at most 9. That is, we assume there are (at least) 9 distinct elements and modify the second disjunct of the conclusion to be that there are 4 distinct elements none of which are R-related. Satallax and Lash both use 186127 steps to find the proof. For Satallax this takes 44 s while for Lash this takes 5.5 s.

The TPTP problem `SYO506^1` is about an if-then-else operator. The problem has a constant c of type $o\iota\iota\iota$. Instead of giving axioms indicating c behaves as an if-then-else operator, the conjecture is given as a disjunction:

$$(\forall xy.c\ (x = y)\ x\ y\ =\ y) \vee \neg(\forall xy.c\ \top\ x\ y\ =\ x) \vee \neg(\forall xy.c\ \bot\ x\ y\ =\ y).$$

After negating the conjecture and applying the first few tableau rules the branch will contain the propositions $\forall xy.c\ \top\ x\ y\ =\ x$, $\forall xy.c\ \bot\ x\ y\ =\ y$ and the disequation $c\ (d = e)\ d\ e \neq e$ for fresh d and e of type ι. In principle the rules for if-then-else given in [2] could be used to solve the problem without using the universally quantified formulas (other than to justify that c *is* an if-then-else operator). However, these are not implemented in Satallax or Lash. Instead search proceeds as usual via the basic underlying procedure. Both Satallax and Lash can prove the example using modes MODE0C1 in 32704 steps. Satallax performs the search in 9.8 s while Lash completes the search in 0.2 s.

In addition to the examples considered above, we have constructed a family of examples intended to demonstrate the power of the shared term representation and caching of operations. Let cons have type $\iota\iota\iota$ and nil have type ι. For each natural number n, consider the proposition C^n given by

$$\overline{n}\ (\lambda x.\mathsf{cons}\ x\ x)\ (\mathsf{cons}\ \mathsf{nil}\ \mathsf{nil}) = \mathsf{cons}\ (\overline{n}\ (\lambda x.\mathsf{cons}\ x\ x)\ \mathsf{nil})\ (\overline{n}\ (\lambda x.\mathsf{cons}\ x\ x)\ \mathsf{nil})$$

where $\overline{n}$ is the appropriately typed Church numeral. Proving the proposition C^n does not require any search and merely requires the prover to normalize the conjecture and note the two sides have the same normal form. However, this normal form on both sides will be a complete binary tree of depth $n+1$. We have run Lash and Satallax on C^n with $n \in \{20, 21, 22, 23, 24\}$ using mode MODE213D. Lash solves all five problems in the same amount of time, less than 0.02 s for each. Satallax takes 4 s, 8 s, 16 s, 32 s and 64 s. As expected, since Satallax is not using a shared representation, the computation time exponentially increases with respect to n.

5 Conclusion and Future Work

We have used Lash as a vehicle to demonstrate that giving a more efficient implementation of the underlying tableau calculus of Satallax can lead to significant performance improvements. An obvious possible extension of Lash would be to implement pattern clauses, higher-order unification and the ability to call E.

While we may do this, our current plans are to focus on directions that further diverge from the development path followed by Satallax.

Interesting theoretical work would be to modify the underlying calculus (while maintaining completeness). For example the rules of the calculus might be able to be further restricted based on orderings of ground terms. On the other hand, new rules might be added to support a variety of constants with special properties. This was already done for constants that satisfy axioms indicating the constant is a choice, description or if-then-else operator [2]. Suppose a constant r of type $\iota\iota o$ is known to be reflexive due to a formula $\forall x.r\ x\ x$ being on the branch. One could avoid ever instantiating this universally quantified formula by simply including a tableau rule that extends a branch with $s \neq t$ whenever $\neg r\ s\ t$ is on the branch. Similar rules could operationalize other special cases of universally quantified formulas, e.g., formulas giving symmetry or transitivity of a relation. A modification of the usual completeness proof would be required to prove completeness of the calculus with these additional rules (and with the restriction disallowing instantiating the corresponding universally quantified formulas).

Finally the C representation of terms could be extended to include precomputed special features. Just as the current implementation knows which de Bruijns are free in the term (without traversing the term), a future implementation could know other features of the term without requiring traversal. Such features could be used to guide the search.

Acknowledgements. The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902 and the ERC starting grant no. 714034 SMART.

References

1. Andrews, P.B., Bishop, M., Issar, S., Nesmith, D., Pfenning, F., Xi, H.: TPS: a theorem-proving system for classical type theory. J. Autom. Reason. **16**(3), 321–353 (1996). https://doi.org/10.1007/BF00252180
2. Backes, J., Brown, C.E.: Analytic tableaux for higher-order logic with choice. J. Autom. Reason. **47**(4), 451–479 (2011). https://doi.org/10.1007/s10817-011-9233-2
3. Bhayat, A., Reger, G.: A combinator-based superposition calculus for higher-order logic. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12166, pp. 278–296. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51074-9_16
4. Brown, C.E.: Satallax: an automatic higher-order prover. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 111–117. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_11
5. Brown, C.E., Smolka, G.: Extended first-order logic. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 164–179. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03359-9_13
6. Brown, C.E., Smolka, G.: Analytic tableaux for simple type theory and its first-order fragment. Logical Methods Comput. Sci. **6**(2) (2010). https://doi.org/10.2168/LMCS-6(2:3)2010

7. Färber, M., Brown, C.: Internal guidance for Satallax. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 349–361. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_24
8. Steen, A., Benzmüller, C.: Extensional higher-order paramodulation in Leo-III. J. Autom. Reason. **65**(6), 775–807 (2021). https://doi.org/10.1007/s10817-021-09588-x
9. Sutcliffe, G.: The TPTP problem library and associated infrastructure. From CNF to TH0, TPTP v6.4.0. J. Autom. Reason. **59**(4), 483–502 (2017)
10. Sutcliffe, G.: The 10th IJCAR automated theorem proving system competition - CASC-J10. AI Commun. **34**(2), 163–177 (2021). https://doi.org/10.3233/AIC-201566
11. Vukmirović, P., Bentkamp, A., Blanchette, J., Cruanes, S., Nummelin, V., Tourret, S.: Making higher-order superposition work. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 415–432. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79876-5_24
12. Vukmirović, P., Blanchette, J.C., Cruanes, S., Schulz, S.: Extending a Brainiac prover to lambda-free higher-order logic. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 192–210. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_11

Goéland: A Concurrent Tableau-Based Theorem Prover (System Description)

Julie Cailler, Johann Rosain, David Delahaye, Simon Robillard(✉), and Hinde Lilia Bouziane

LIRMM, Univ Montpellier, CNRS, Montpellier, France
{julie.cailler,johann.rosain,david.delahaye,simon.robillard,
hinde.bouziane}@lirmm.fr

Abstract. We describe Goéland, an automated theorem prover for first-order logic that relies on a concurrent search procedure to find tableau proofs, with concurrent processes corresponding to individual branches of the tableau. Since branch closure may require instantiating free variables shared across branches, processes communicate via channels to exchange information about substitutions used for closure. We present the proof search procedure and its implementation, as well as experimental results obtained on problems from the TPTP library.

Keywords: Automated Theorem Proving · Tableaux · Concurrency

1 Introduction

Although clausal proof techniques have enjoyed success in automated theorem proving, some applications benefit from reasoning on unaltered formulas (rather than Skolemized clauses), while others require the production of proofs in a sequent calculus. These roles are fulfilled by provers based on the tableau method [17], as initially designed by Beth and Hintikka [2,13]. For first-order logic, efficient handling of universal formulas is typically achieved with free variables that are instantiated only when needed to close a branch. This step is said to be *destructive* because it may affect open branches sharing variables. This causes fairness (and consequently, completeness) issues, as illustrated in Fig. 1. In this example, exploring the left branch produces a substitution that prevents direct closure of the right branch. Reintroducing the original quantified formula with a different free variable is not sufficient to close the right branch, because an applicable δ-rule creates a new Skolem symbol that will result in a different but equally problematic substitution every time a left branch is explored. Thus, systematically exploring the left branch before the right leads to non-termination of the search. Conversely, exploring the right branch first produces a substitution (which instantiates the free variable X with a rather than b) that closes both branches.

Concurrent computing offers a way to implement a proof search procedure that explores branches simultaneously. Such a procedure can compare closing

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 359–368, 2022.
https://doi.org/10.1007/978-3-031-10769-6_22

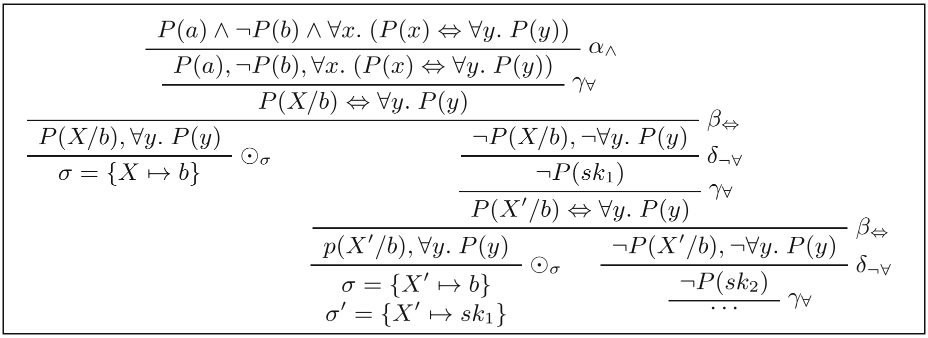

$P(a) \wedge \neg P(b) \wedge \forall x.\ (P(x) \Leftrightarrow \forall y.\ P(y))$
$\alpha_\wedge$
$P(a), \neg P(b), \forall x.\ (P(x) \Leftrightarrow \forall y.\ P(y))$
$\gamma_\forall$
$P(X/b) \Leftrightarrow \forall y.\ P(y)$
$\beta_\Leftrightarrow$
$P(X/b), \forall y.\ P(y)$
$\odot_\sigma$
$\sigma = \{X \mapsto b\}$
$\neg P(X/b), \neg\forall y.\ P(y)$
$\delta_{\neg\forall}$
$\neg P(sk_1)$
$\gamma_\forall$
$P(X'/b) \Leftrightarrow \forall y.\ P(y)$
$\beta_\Leftrightarrow$
$p(X'/b), \forall y.\ P(y)$
$\odot_\sigma$
$\sigma = \{X' \mapsto b\}$
$\sigma' = \{X' \mapsto sk_1\}$
$\neg P(X'/b), \neg\forall y.\ P(y)$
$\delta_{\neg\forall}$
$\neg P(sk_2)$
$\gamma_\forall$
$\ldots$

Fig. 1. Incompleteness caused by unfair selection of branches

substitutions to detect (dis)agreements between branches, and consequently either close branches early, or restart proof attempts with limited backtracking. The simultaneous exploration of branches is handled by the concurrency system, either by interleaving computations through scheduling, or by executing tasks in parallel if the hardware resources allow it. A concurrent procedure naturally lends itself to parallel execution, allowing us to take advantage of multi-core architectures for efficient first-order theorem proving. Thus, concurrency provides an elegant and efficient solution to proof search with free variable tableaux.

In this paper, we describe a concurrent destructive proof search procedure for first-order analytic tableaux (Sect. 2) and its implementation in a tool called Goéland, as well as its evaluation on problems from the TPTP library [19] and comparison to other state-of-the-art provers (Sect. 3).

Related Work. A lot of research has been carried out on the parallelization of proof search procedures [4], often focusing primarily on parallel execution and performance. In contrast, we use concurrency not only as a way to take advantage of multi-core architectures, but also as an algorithmic device that is useful even for sequential execution (with interleaved threads). Some concurrent and parallel approaches focus more distinctly on the exploration of the search space, either by dividing the search space between processes (*distributed search*) or by using processes with different search plans on the same space (*multi search*) [3]. These approaches can be performed either by *heterogeneous systems* that rely on cooperation between systems with different inference systems [1,8,12], or *homogeneous systems* where all deductive processes use the same inference system. According to this classification, the technique presented here is a homogeneous system that performs a distributed search. Concurrent tableaux provers include the model-elimination provers CPTheo [12] and Partheo [18], and the higher-order prover Hot [15], which notably uses concurrency to deal with fairness issues arising from the non-terminating nature of higher-order unification. Lastly, concurrency has been used as the basis of a generic framework to present various proof strategies [10] or allow distributed calculations over a network [21].

2 Concurrent Proof Search

Free Variable Tableaux. Goéland attempts to build a refutation proof for a first-order formula, i.e., a closed tableau for its negation, using a standard free-variable tableau calculus [11]. The calculus is composed of α-, γ- and δ-rules that extend a branch with one formula, β-rules that divide a branch by extending it with two formulas, and a $\odot$-rule that closes a branch. γ-rules deal with universally-quantified formulas by introducing a formula with a free variable. A free variable is not universally quantified, but is instead a placeholder for some term instantiation, typically determined upon branch closure. δ-rules deal with existentially-quantified formulas by introducing a formula with a Skolem function symbol that takes as arguments the free variables in the branch. This ensures freshness of the Skolem symbol independently of variable instantiation.

The branch closure rule applies to a branch carrying atomic formulas P and Q such that, for some substitution σ, $\sigma(P) = \sigma(\neg Q)$. In that case, σ is applied to all branches. That rule is consequently *destructive*: applying a substitution to close one branch may modify another, removing the possibility to close it immediately. A tableau is closed when all its branches are closed. Closing a tableau can thus be seen as providing a global unifier that closes all branches.

Semantics for Concurrency. Goéland relies on a concurrent search procedure. In order to present this procedure, we use a simple WHILE language augmented with instructions for concurrency, in the style of CSP [14]. Each process has its own variable store, as well as a collection of process identifiers used for communication: π_{parent} denotes the identifier of a process's parent, while Π_{children} denotes the collection of identifiers of active children of that process. Given a process identifier π and an expression e, the command $\pi\,\mathbf{!}\,e$ is used to send an asynchronous message with the value e to the process identified by π. Conversely, the command $\pi\,\mathbf{?}\,x$ blocks the execution until the process identified by π sends a message, which is stored in the variable x. Lastly, the instruction **start** creates a new process that executes a function with some given arguments, while the instruction **kill** interrupts the execution of a process according to its identifier.

Proof Search Procedure. The proof search is carried out concurrently by processes corresponding to branches of the tableau. Processes are started upon application of a β-rule, one for each new branch. Communications between processes take two forms: a process may send a set of closing substitutions for its branch to its parent, or a parent may send a substitution (that closes one of its children's branch) to the other children. The proof search is performed by the *proofSearch*, *waitForParent*, and *waitForChildren* procedures (described in Procedures 1, 2, and 3, respectively).

The *proofSearch* procedure initiates the proof search for a branch. It first attempts to apply the closure rule. A closing substitution is called *local* to a process if its domain includes only free variables introduced by this process or one of its descendants (i.e., if the variables do not occur higher in the proof tree). If one of the closing substitutions is local to the process, it is reported and the

Procedure 1: *proofSearch*

```
Data: a tableau $T$
begin
    var $\Theta \leftarrow applyClosingRule(T)$ ;
    for $\theta \in \Theta$ do
        if $isLocal(\theta)$ then
            $\pi_{\text{parent}}$ ! $\{\theta\}$
            return
    if $\Theta \neq \emptyset$ then
        $\pi_{\text{parent}}$ ! $\Theta$
        $waitForParent(T, \Theta)$
    else if $applicableAlphaRule(T)$ then
        $proofSearch(applyAlphaRule(T))$
    else if $applicableDeltaRule(T)$ then
        $proofSearch(applyDeltaRule(T))$
    else if $applicableBetaRule(T)$ then
        for $T' \in applyBetaRule(T)$ do
            start $proofSearch(T')$
        $waitForChildren(T, \emptyset, \emptyset)$
    else if $applicableGammaRule(T)$ then
        $proofSearch(applyGammaRule(T))$
    else
        $\pi_{\text{parent}}$ ! $\emptyset$
```

process terminates. If only non-local closing substitutions are found, they are reported and the process executes *waitForParent*. Otherwise, the procedure applies tableau expansion rules according to the priority: $\alpha \prec \delta \prec \beta \prec \gamma$. If a β-rule is applied, new processes are started, and each of them executes *proofSearch* on the newly created branch, while the current process executes *waitForChildren*.

The *waitForParent* procedure is executed by a process after it has found closing non-local substitutions. Such substitutions may prevent closure in other branches. In these cases, the parent will eventually send another candidate substitution. *waitForParent* waits until such a substitution is received, and triggers a new step of proof search. The process may also be terminated by its parent (via the **kill** instruction) during the execution of this procedure, if one of the substitutions previously sent by the process leads to closing the parent's branch.

The *waitForChildren* procedure is executed by a process after the application of a β-rule and the creation of child processes. The set of substitutions sent by each child is stored in a map *subst* (Line 2), initially undefined everywhere ($f_{\perp}$). This procedure closes the branch (Line 13) if there exists a substitution θ that agrees with one closing substitution of each child process, i.e., for each child process, the process has reported a substitution σ such that $\sigma(X) = \theta(X)$ for any variable X in the domain of σ. If no such substitution can be found

Procedure 2: *waitForParent*

```
  Data: a tableau $T$, a set $\Theta_{\text{sent}}$ of substitutions sent by this process to its parent
1 begin
2     $\pi_{\text{parent}}$ ? $\sigma$
3     if $\sigma \in \Theta_{sent}$ then
4         $\pi_{\text{parent}}$ ! $\sigma$
5         $waitForParent(T, \Theta_{\text{sent}})$
6     else
7         $proofSearch(\sigma(T))$
```

after all the children have closed their branches, then one closing substitution $\sigma \in$ *subst* is picked arbitrarily (Line 18) and sent to all the children (which are at that point executing *waitForParent*) to restart their proof attempts. With the additional constraint of the substitution σ, the new proof attempts may fail, hence the necessity for backtracking among candidate substitutions $\Theta_{\text{backtrack}}$ (Line 5 and 6). At the end, if all the substitutions were tried and failed, the process sends a failure message (symbolized by $\emptyset$) to its parent.

Thus, concurrency and backtracking are used to prevent incompleteness resulting from unfair instantiation of free variables. Another potential source of unfairness is the γ-rule, when applied more than once to a universal formula (reintroduction). This may be needed to find a refutation, but unbounded reintroductions would lead to unfairness. Iterative deepening [16] is used to guard against this: a bound limits the number of reintroductions on any single branch, and if no proof is found, the bound is increased and the proof search restarted.

Figure 2 illustrates the interactions between processes for the problem in Fig. 1, and shows how concurrency helps ensure fairness. It describes the parent process, in the top box, and below, the two children processes created upon application of the β-rule. Dotted lines separate successive states of a process (i.e., Procedures 1, 2 and 3 seen above), while arrows and boxes represent substitution exchanges. The number above each arrow indicates the chronology of the interactions. After both children have returned a substitution (1), the parent arbitrarily chooses one of them, starting with $X \mapsto b$, and sends it to the children (2). Since this substitution prevents closure in the right branch (3), the parent later backtracks and sends the other substitution $X \mapsto a$ (4), allowing both children (5) and then the parent to close successfully.

3 Implementation and Experimental Results

Implementation. The procedures presented in Sect. 2 are implemented in the Goéland prover[1] using the Go language. Go supports concurrency and parallelism, based on lightweight execution threads called *goroutines* [20]. Goroutines

[1] Available at: https://github.com/GoelandProver/Goeland/releases/tag/v1.0.0-beta.

Procedure 3: *waitForChildren*

```
Data: a tableau $T$, a set $\Theta_{\text{sent}}$ of substitutions sent by this process to its
      parent, a set $\Theta_{\text{backtrack}}$ of substitutions used for backtracking
begin
    var subst $\leftarrow \mathrm{f}_{\perp}$
    while $\exists \pi \in \mathit{\Pi_{children}}.\, \mathrm{subst}[\pi] = \perp$ do
        $\pi$ ? subst[$\pi$]
        if subst[$\pi$] $= \emptyset$ then
            if $\exists \theta \in \Theta_{backtrack}$ then
                for $\pi \in \mathit{\Pi_{children}}$ do $\pi$ ! $\theta$;
                $waitForChildren(T, \Theta_{\text{sent}}, \Theta_{\text{backtrack}} \setminus \{\theta\})$
            else
                for $\pi \in \mathit{\Pi_{children}}$ do kill $\pi$;
                $\pi_{\text{parent}}$ ! $\emptyset$
                return
    if $\exists \theta, \mathrm{agreement}(\theta, \mathrm{subst})$ then
        $\pi_{\text{parent}}$ ! $\{\theta\}$
        for $\pi \in \mathit{\Pi_{children}}$ do kill $\pi$;
        $waitForParent(T, \Theta_{\text{sent}} \cup \{\theta\})$
    else
        $\sigma \leftarrow \mathrm{choice}(\mathrm{subst})$
        for $\pi \in \mathit{\Pi_{children}}$ do $\pi$ ! $\sigma$;
        $waitForChildren(T, \Theta_{\text{sent}}, \Theta_{\text{backtrack}} \cup \bigcup_{\pi} \mathrm{subst}[\pi] \setminus \{\sigma\}))$
```

are executed according to a so-called *hybrid threading* (or $M : N$) model: M goroutines are executed over N effective threads and scheduling is managed by both the Go runtime and the operating system. This threading model allows the execution of a large number of goroutines with a reasonable consumption of system resources. Goroutines use channels to exchange messages, so that the implementation is close to the presentation of Sect. 2.

Goéland has, for the time being, no dedicated mechanism for equality reasoning. However, we have implemented an extension that implements deduction modulo theory [9], i.e., transforms axioms into rewrite rules over propositions and terms. Deduction modulo theory has proved very useful to improve proof search when integrated into usual automated proof techniques [5], and also produces excellent results with manually-defined rewrite rules [6,7]. In Goéland, deduction modulo theory selects some axioms on the basis of a simple syntactic criterion and replaces them by rewrite rules.

Experimental Results. We evaluated Goéland on two problems categories with FOF theorems in the TPTP library (v7.4.0): syntactic problems without equality (SYN) and problems of set theory (SET). The former was chosen for its elementary nature, whereas the latter was picked primarily to evaluate the performance of the deduction modulo theory, as the axioms of set theory are good

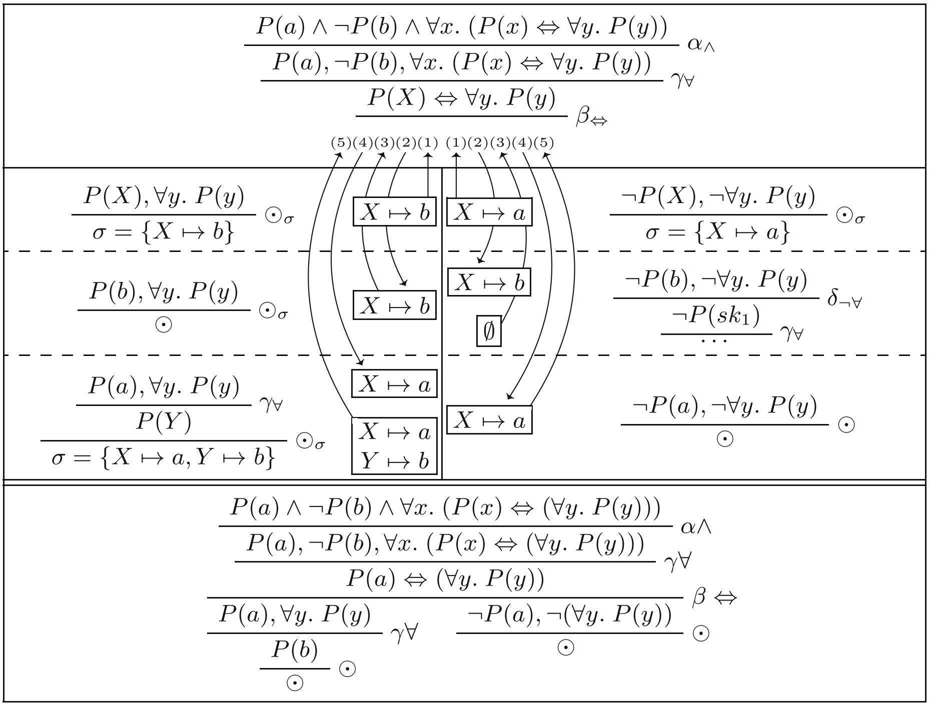

$P(a) \wedge \neg P(b) \wedge \forall x.\ (P(x) \Leftrightarrow \forall y.\ P(y))$
$\alpha_\wedge$
$P(a), \neg P(b), \forall x.\ (P(x) \Leftrightarrow \forall y.\ P(y))$
$\gamma_\forall$
$P(X) \Leftrightarrow \forall y.\ P(y)$
$\beta_\Leftrightarrow$
(5)(4)(3)(2)(1) (1)(2)(3)(4)(5)
$P(X), \forall y.\ P(y)$
$\sigma = \{X \mapsto b\}$
$\odot_\sigma$
$X \mapsto b$
$X \mapsto a$
$\neg P(X), \neg \forall y.\ P(y)$
$\sigma = \{X \mapsto a\}$
$\odot_\sigma$
$P(b), \forall y.\ P(y)$
$\odot$
$\odot_\sigma$
$X \mapsto b$
$X \mapsto b$
$\emptyset$
$\neg P(b), \neg \forall y.\ P(y)$
$\delta_{\neg\forall}$
$\neg P(sk_1)$
$\dots$
$\gamma_\forall$
$P(a), \forall y.\ P(y)$
$\gamma_\forall$
$P(Y)$
$\odot_\sigma$
$\sigma = \{X \mapsto a, Y \mapsto b\}$
$X \mapsto a$
$X \mapsto a$
$Y \mapsto b$
$X \mapsto a$
$\neg P(a), \neg \forall y.\ P(y)$
$\odot$
$\odot$
$P(a) \wedge \neg P(b) \wedge \forall x.\ (P(x) \Leftrightarrow (\forall y.\ P(y)))$
$\alpha\wedge$
$P(a), \neg P(b), \forall x.\ (P(x) \Leftrightarrow (\forall y.\ P(y)))$
$\gamma\forall$
$P(a) \Leftrightarrow (\forall y.\ P(y))$
$\beta \Leftrightarrow$
$P(a), \forall y.\ P(y)$
$\gamma\forall$
$P(b)$
$\odot$
$\odot$
$\neg P(a), \neg(\forall y.\ P(y))$
$\odot$
$\odot$

Fig. 2. Proof search and resulting proof for $P(a) \wedge \neg P(b) \wedge \forall x.(P(x) \Leftrightarrow \forall y.P(y))$

targets for rewriting. We compared the results with those of five other provers: tableau-based provers Zenon (v0.8.5), Princess (v2021-05-10) and LeoIII (v1.6), as well as saturation-based provers E (v2.6) and Vampire (v4.6.1). Experiments were executed on a computer equipped with an Intel Xeon E5-2680 v4 2.4GHz 2×14-core processor and 128 GB of memory. Each proof attempt was limited to 300 s. Table 1 and Fig. 3 report the results. Table 1 shows the number of problems solved by each prover, the cumulative time, and the number of problems solved by a given prover but not by Goéland (+) and conversely (−). Figure 3 presents the cumulative time required to solve the number of problems.

As can be observed, the results of Goéland are comparable to, or slightly better than those of other tableau-based provers on problems from SYN, while saturation theorem provers achieve the best results. On this category, the axioms do not trigger deduction modulo theory rewriting rules, hence the similar results of Goéland and Goéland+DMT. On SET, Goéland+DMT obtains significantly better results than other tableau-based provers. This confirms the previous results on the performance of deduction modulo theory for set theory [6,7].

Table 1. Experimental results over the TPTP library

	SYN (263 problems)	SET (464 problems)
Goéland	**199** (190 s)	**150** (4659 s)
Goéland+DMT	**199** (196 s) (+0, −0)	**278** (1292 s) (+142, −14)
Zenon	**256** (67 s) (+60, −3)	**150** (562 s) (+75, −75)
Princess	**195** (189 s) (+1, −5)	**258** (1168 s) (+141, −33)
LeoIII	**195** (268 s) (+1, −5)	**177** (2925 s) (+77, −50)
E	**261** (168 s) (+62, −0)	**363** (2377 s) (+223, −10)
Vampire	**262** (13 s) (+63, −0)	**321** (4122 s) (+188, −17)

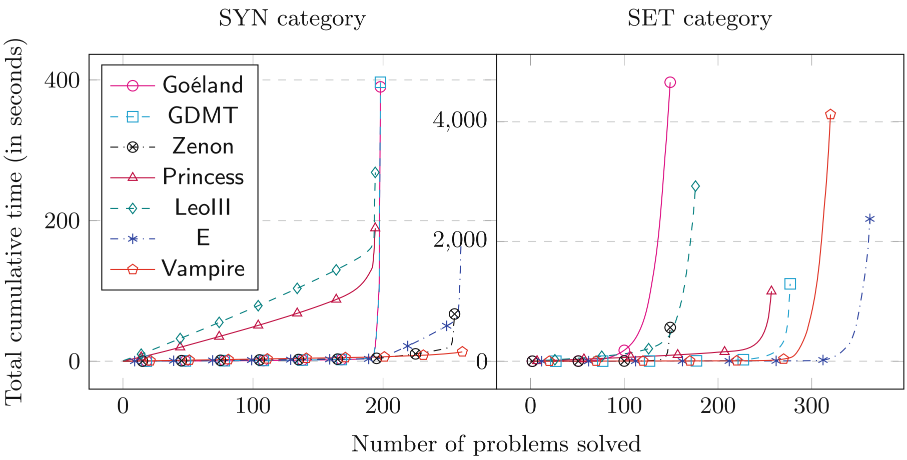


Fig. 3. Cumulative time per problem solved between Goéland, Goéland+DMT(GDMT), Zenon, Princess, LeoIII, E, and Vampire

4 Conclusion

We have presented a concurrent proof search procedure for tableaux in first-order logic with the aim of ensuring a fair exploration of the search space. This procedure has been implemented in the prover Goéland. This tool is still in an early stage, and (with the exception of deduction modulo theory) implements only the most basic functionalities, yet empirical results are encouraging. We plan on adding functionalities such as equality reasoning, arithmetic reasoning, and support for polymorphism to Goéland, which should increase its usability and performance. The integration of these functionalities in the context of a concurrent prover seems to be a promising line of research. Further investigation is also needed to prove the fairness, and therefore completeness, of our procedure.

References

1. Benzmüller, C., Kerber, M., Jamnik, M., Sorge, V.: Experiments with an agent-oriented reasoning system. In: Baader, F., Brewka, G., Eiter, T. (eds.) KI 2001. LNCS (LNAI), vol. 2174, pp. 409–424. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45422-5_29
2. Beth, E.W.: Formal Methods: An Introduction to Symbolic Logic and to the Study of Effective Operations in Arithmetic and Logic, Synthese Library, vol. 4. D. Reidel Pub. Co. (1962)
3. Bonacina, M.P.: A taxonomy of parallel strategies for deduction. Ann. Math. Artif. Intell. **29**(1), 223–257 (2000)
4. Bonacina, M.P.: Parallel theorem proving. In: Hamadi, Y., Sais, L. (eds.) Handbook of Parallel Constraint Reasoning, pp. 179–235. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-63516-3_6
5. Burel, G., Bury, G., Cauderlier, R., Delahaye, D., Halmagrand, P., Hermant, O.: First-order automated reasoning with theories: when deduction modulo theory meets practice. J. Autom. Reason. **64**(6), 1001–1050 (2019). https://doi.org/10.1007/s10817-019-09533-z
6. Bury, G., Cruanes, S., Delahaye, D., Euvrard, P.-L.: An automation-friendly set theory for the B method. In: Butler, M., Raschke, A., Hoang, T.S., Reichl, K. (eds.) ABZ 2018. LNCS, vol. 10817, pp. 409–414. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91271-4_32
7. Bury, G., Delahaye, D., Doligez, D., Halmagrand, P., Hermant, O.: Automated deduction in the B set theory using typed proof search and deduction modulo. In: Fehnker, A., McIver, A., Sutcliffe, G., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence and Reasoning (LPAR). EPiC Series in Computing, vol. 35, pp. 42–58. EasyChair (2015)
8. Denzinger, J., Kronenburg, M., Schulz, S.: DISCOUNT - a distributed and learning equational prover. J. Autom. Reason. **18**(2), 189–198 (1997)
9. Dowek, G., Hardin, T., Kirchner, C.: Theorem proving modulo. J. Autom. Reason. (JAR) **31**(1), 33–72 (2003)
10. Fisher, M.: An open approach to concurrent theorem proving. Parallel Process. Artif. Intell. **3**, 80011 (1997)
11. Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer, Heidelberg (1990)
12. Fuchs, M., Wolf, A.: System description: cooperation in model elimination: CPTHEO. In: Kirchner, C., Kirchner, H. (eds.) CADE 1998. LNCS, vol. 1421, pp. 42–46. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054245
13. Hintikka, J.: Two papers on symbolic logic: form and content in quantification theory and reductions in the theory of types. Societas Philosophica, Acta philosophica Fennica **8**, 7–55 (1955)
14. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**(8), 666–677 (1978)
15. Konrad, K.: Hot: a concurrent automated theorem prover based on higher-order tableaux. In: Grundy, J., Newey, M. (eds.) TPHOLs 1998. LNCS, vol. 1479, pp. 245–261. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055140
16. Korf, R.E.: Depth-first iterative-deepening: an optimal admissible tree search. Artif. Intell. **27**(1), 97–109 (1985)

17. Letz, R.: First-order tableau methods. In: D'Agostino, M., Gabbay, D.M., Hähnle, R., Posegga, J. (eds.) Handbook of Tableau Methods, pp. 125–196. Springer, Heidelberg (1999). https://doi.org/10.1007/978-94-017-1754-0_3. ISBN 978-94-017-1754-0
18. Schumann, J., Letz, R.: Partheo: a high-performance parallel theorem prover. In: Stickel, M.E. (ed.) CADE 1990. LNCS, vol. 449, pp. 40–56. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52885-7_78
19. Sutcliffe, G.: The TPTP problem library and associated infrastructure. From CNF to TH0, TPTP v6.4.0. J. Autom. Reason. (JAR) **59**(4), 483–502 (2017)
20. Tsoukalos, M.: Mastering Go: Create Golang Production Applications Using Network Libraries, Concurrency, Machine Learning, and Advanced Data Structures, pp. 439–463. Packt Publishing Ltd. (2019)
21. Wu, C.H.: A multi-agent framework for distributed theorem proving. Expert Syst. Appl. **29**(3), 554–565 (2005)

Binary Codes that Do Not Preserve Primitivity

Štěpán Holub[1](✉), Martin Raška[1], and Štěpán Starosta[2](✉)

[1] Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic
holub@karlin.mff.cuni.cz

[2] Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic
stepan.starosta@fit.cvut.cz

Abstract. A code X is not primitivity preserving if there is a primitive list $\mathbf{w} \in \texttt{lists}\, X$ whose concatenation is imprimitive. We formalize a full characterization of such codes in the binary case in the proof assistant Isabelle/HOL. Part of the formalization, interesting on its own, is a description of $\{x, y\}$-interpretations of the square xx if $|y| \leq |x|$. We also provide a formalized parametric solution of the related equation $x^j y^k = z^\ell$.

1 Introduction

Consider two words `abba` and `b`. It is possible to concatenate (several copies of) them as `b`·`abba`·`b`, and obtain a power of a third word, namely a square `bab`·`bab` of `bab`. In this paper, we completely describe all ways how this can happen for two words, and formalize it in Isabelle/HOL.

The corresponding theory has a long history. The question can be formulated as solving equations in three variables of the special form $W(x, y) = z^\ell$ where the left hand side is a sequence of x's and y's, and $\ell \geq 2$. The seminal result in this direction is the paper by R. C. Lyndon and M.-P. Schützenberger [10] from 1962, which solves in a more general setting of free groups the equation $x^j y^k = z^\ell$ with $2 \leq j, k, \ell$. It was followed, in 1967, by a partial answer to our question by A. Lentin and M.-P. Schützenberger [9]. A complete characterization of monoids generated by three words was provided by L. G. Budkina and Al. A. Markov in 1973 [4]. The characterization was later, in 1976, reproved in a different way by Lentin's student J.-P. Spehner in his Ph.D. thesis [14], which even explicitly mentions the answer to the present question. See also a comparison of the two classifications by T. Harju and D. Nowotka [7]. In 1985, the result was again reproved by E. Barbin-Le Rest and M. Le Rest [1], this time specifically focusing on our question. Their paper contains a characterization of binary interpretations of a square as a crucial tool. The latter combinatorial result is interesting on its own, but is very little known. In addition to the fact that, as far as we know, the proof is not available in English, it has to be reconstructed from Théorème 2.1 and Lemme 3.1 in [1], it is long, technical and little structured, with many

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 369–387, 2022.
https://doi.org/10.1007/978-3-031-10769-6_23

intuitive steps that have to be clarified. It is symptomatic, for example, that Maňuch [11] cites the claim as essentially equivalent to his desired result but nevertheless provides a different, shorter but similarly technical proof.

The fact that several authors opted to provide their own proof of the already known result, and that even a weaker result was republished as new shows that the existing proof was not considered sufficiently convincing and approachable. This makes the topic a perfect candidate for formalization. The proof we present here naturally contains some ideas of the proof from [1] but is significantly different. Our main objective was to follow the basic methodological requirement of a good formalization, namely to identify claims that are needed in the proof and formulate them as separate lemmas and as generally as possible so that they can be reused not only in the proof but also later. Moreover, the formalization naturally forced us to consider carefully the overall strategy of the proof (which is rather lost behind technical details of published works on this topic). Under Isabelle's pressure we eventually arrived at a hopefully clear proof structure which includes a simple, but probably innovative use of the idea of "gluing" words. The analysis of the proof is therefore another, and we believe the most important contribution of our formalization, in addition to the mere certainty that there are no gaps in the proof.

In addition, we provide a complete parametric solution of the equation $x^k y^j = z^\ell$ for arbitrary j, k and ℓ, a classification which is not very difficult, but maybe too complicated to be useful in a mere unverified paper form.

The formalization presented here is an organic part of a larger project of formalization of combinatorics of words (see an introductory description in [8]). We are not aware of a similar formalization project in any proof assistant. The existence of the underlying library, which in turn extends the theories of "List" and "HOL-Library.Sublist" from the standard Isabelle distribution, critically contributes to a smooth formalization which is getting fairly close to the way a human paper proof would look like, outsourcing technicalities to the (reusable) background. We accompany claims in this text with names of their formalized counterparts.

2 Basic Facts and Notation

Let Σ be an arbitrary set. Lists (i.e. finite sequences) $[x_1, x_2, \ldots, x_n]$ of elements $x_i \in \Sigma$ are called *words* over Σ. The set of all words over Σ is usually denoted as Σ^*, using the Kleene star. A notorious ambivalence of this notation is related to the situation when we consider a set of words $X \subset \Sigma^*$, and are interested in lists over X. They should be denoted as elements of X^*. However, X^* usually means something else (in the theory of rational languages), namely the set of all words in Σ^* generated by the set X. To avoid the confusion, we will therefore follow the notation used in the formalization in Isabelle, and write `lists` X instead, to make clear that the entries of an element of `lists` X are themselves words. In order to further help to distinguish words over the basic alphabet from lists over a set of words, we shall use boldface variables for the latter.

In particular, it is important to keep in mind the difference between a letter a and the word $[a]$ of length one, the distinction which is usually glossed over lightly in the literature on combinatorics on words. The set of words over Σ generated by X is then denoted as $\langle X \rangle$. The (associative) binary operation of concatenation of two words u and v is denoted by $u \cdot v$. We prefer this algebraic notation to the Isabelle's original `@`. Moreover, we shall often omit the dot as usual. If $\mathbf{u} = [x_1, x_2, \ldots, x_n] \in \texttt{lists}\, X$ is a list of words, then we write $\texttt{concat}\, \mathbf{u}$ for $x_1 \cdot x_2 \cdots x_n$. We write ε for the empty list, and u^k for the concatenation of k copies of u (we use $u^{\texttt{@}}k$ in the formalization). We write $u \leq_p v$, $u <_p v$, $u \leq_s v$, $u <_s v$, and $u \leq_f v$ to denote that u is a *prefix*, a *strict prefix*, *suffix*, *strict suffix* and *factor* (that is, a contiguous sublist) respectively. A word is *primitive* if it is nonempty and not a power of a shorter word. Otherwise, we call it *imprimitive*. Each nonempty word w is a power of a unique primitive word $\rho\, w$, its *primitive root*. A nonempty word r is a *periodic root* of a word w if $w \leq_p r \cdot w$. This is equivalent to w being a prefix of the right infinite power of r, denoted r^ω. Note that we deal with finite words only, and we use the notation r^ω only as a convenient shortcut for "a sufficiently long power of r". Two words u and v are *conjugate*, we write $u \sim v$, if $u = rq$ and $v = qr$ for some words r and q. Note that conjugation is an equivalence whose classes are also called *cyclic words*. A word u is a *cyclic factor* of w if it is a factor of some conjugate of w. A set of words X is a *code* if its elements do not satisfy any nontrivial relation, that is, they are a basis of a free semigroup. For a two-element set $\{x, y\}$, this is equivalent to x and y being non-commuting, i.e., $xy \neq yx$, and/or to $\rho\, x \neq \rho\, y$. An important characterization of a semigroup S of words to be free is the *stability condition* which is the implication $u, v, uz, zv \in S \Longrightarrow z \in S$. The longest common prefix of u and v is denoted by $u \wedge_p v$. If $\{x, y\}$ is a (binary) code, then $(x \cdot w) \wedge_p (y \cdot w') = xy \wedge_p yx$ for any $w, w' \in \langle\{x, y\}\rangle$ sufficiently long. We explain some elementary facts from combinatorics on words used in this article in more detail in Sect. 8.

3 Main Theorem

Let us introduce the central definition of the paper.

Definition 1. *We say that a set X of words is* primitivity preserving *if there is no word $\mathbf{w} \in \texttt{lists}\, X$ such that*

- $|\mathbf{w}| \geq 2$*;*
- $\mathbf{w}$ *is primitive; and*
- $\texttt{concat}\, \mathbf{w}$ *is imprimitive.*

Note that our definition does not take into account singletons $\mathbf{w} = [x]$. In particular, X can be primitivity preserving even if some $x \in X$ is imprimitive. Nevertheless, in the binary case, we will also provide some information about the cases when one or both elements of the code have to be primitive.

In [12], V. Mitrana formulates the primitivity of a set in terms of morphisms, and shows that X is primitivity preserving if and only if it is the minimal set of generators of a "pure monoid", cf. [3, p. 276]. This brings about a wider concept of morphisms preserving a given property, most classically square-freeness, see for example a characterization of square-free morphisms over three letters by M. Crochemore [5].

The target claim of our formalization is the following characterization of words witnessing that a binary code is not primitivity preserving:

Theorem 1 (`bin_imprim_code`). *Let $B = \{x, y\}$ be a code that is not primitivity preserving. Then there are integers $j \geq 1$ and $k \geq 1$, with $k = 1$ or $j = 1$, such that the following conditions are equivalent for any $\mathbf{w} \in$ `lists` B with $|\mathbf{w}| \geq 2$:*

- *$\mathbf{w}$ is primitive, and `concat` $\mathbf{w}$ is imprimitive*
- *$\mathbf{w}$ is conjugate with $[x]^j[y]^k$.*

Moreover, assuming $|y| \leq |x|$,

- *if $j \geq 2$, then $j = 2$ and $k = 1$, and both x and y are primitive;*
- *if $k \geq 2$, then $j = 1$ and x is primitive.*

Proof. Let $\mathbf{w}$ be a word witnessing that B is not primitivity preserving. That is, $|\mathbf{w}| \geq 2$, $\mathbf{w}$ is primitive, and `concat` $\mathbf{w}$ is imprimitive. Since $[x]^j[y]^k$ and $[y]^k[x]^j$ are conjugate, we can suppose, without loss of generality, that $|y| \leq |x|$.

First, we want to show that $\mathbf{w}$ is conjugate with $[x]^j[y]^k$ for some $j, k \geq 1$ such that $k = 1$ or $j = 1$. Since $\mathbf{w}$ is primitive and of length at least two, it contains both x and y. If it contains one of these letters exactly once, then $\mathbf{w}$ is clearly conjugate with $[x]^j[y]^k$ for $j = 1$ or $k = 1$. Therefore, the difficult part is to show that no primitive $\mathbf{w}$ with `concat` $\mathbf{w}$ imprimitive can contain both letters at least twice. This is the main task of the rest of the paper, which is finally accomplished by Theorem 4 claiming that words that contain at least two occurrences of x are conjugate with $[x, x, y]$. To complete the proof of the first part of the theorem, it remains to show that j and k do not depend on $\mathbf{w}$. This follows from Lemma 1.

Note that the imprimitivity of `concat` $\mathbf{w}$ induces the equality $x^j y^k = z^\ell$ for some z and $\ell \geq 2$. The already mentioned seminal result of Lyndon and Schützenberger shows that j and k cannot be simultaneously at least two, since otherwise x and y commute. For the same reason, considering its primitive root, the word y is primitive if $j \geq 2$. Similarly, x is primitive if $k \geq 2$. The primitivity of x when $j = 2$ is a part of Theorem 4. □

We start by giving a complete parametric solution of the equation $x^j y^k = z^\ell$ in the following theorem. This will eventually yield, after the proof of Theorem 1 is completed, a full description of not primitivity preserving binary codes. Since the equation is mirror symmetric, we omit symmetric cases by assuming $|y| \leq |x|$.

Theorem 2 (`LS_parametric_solution`). *Let $\ell \geq 2$, $j, k \geq 1$ and $|y| \leq |x|$.*

The equality $x^j y^k = z^\ell$ holds if and only if one of the following cases takes place:

A. There exists a word r, and integers $m, n, t \geq 0$ such that

$$mj + nk = t\ell, \quad \text{and}$$
$$x = r^m, \quad y = r^n, \quad z = r^t;$$

B. $j = k = 1$ and there exist non-commuting words r and q, and integers $m, n \geq 0$ such that

$$m + n + 1 = \ell, \quad \text{and}$$
$$x = (rq)^m r, \quad y = q(rq)^n, \quad z = rq;$$

C. $j = \ell = 2$, $k = 1$ and there exist non-commuting words r and q and an integer $m \geq 2$ such that

$$x = (rq)^m r, \quad y = qrrq, \quad z = (rq)^m rrq;$$

D. $j = 1$ and $k \geq 2$ and there exist non-commuting words r and q such that

$$x = (qr^k)^{\ell-1} q, \quad y = r, \quad z = qr^k;$$

E. $j = 1$ and $k \geq 2$ and there are non-commuting words r and q, an integer $m \geq 1$ such that

$$x = (qr(r(qr)^m)^{k-1})^{\ell-2} qr(r(qr)^m)^{k-2} rq, \quad y = r(qr)^m, \quad z = qr(r(qr)^m)^{k-1}.$$

Proof. If x and y commute, then all three words commute, hence they are a power of a common word. A length argument yields the solution A.

Assume now that $\{x, y\}$ is a code. Then no pair of words x, y and z commutes. We have shown in the overview of the proof of Theorem 1 that $j = 1$ or $k = 1$ by the Lyndon-Schützenberger theorem. The solution is then split into several cases.

Case 1: $j = k = 1$.
Let m and r be such that $z^m r = x$ with r a strict prefix of z. By setting $z = rq$, we obtain the solution B with $n = \ell - m - 1$.

Case 2: $j \geq 2, k = 1$.
Since $|y| \leq |x|$ and $\ell \geq 2$, we have

$$2|z| \leq |z^\ell| = |x^j| + |y| < 2|x^j|,$$

so z is a strict prefix of x^j.

As x^j has periodic roots both z and x, and z does not commute with x, the Periodicity lemma implies $|x^j| < |z| + |x|$. That is, $z = x^{j-1}u$, $x^j = zv$ and $x = uv$ for some nonempty words u and v. As v is a prefix of z, it is also a prefix of x. Therefore, we have

$$x = uv = vu'$$

for some word u'. This is a well known conjugation equality which implies $u = rq$, $u' = qr$ and $v = (rq)^n r$ for some words r, q and an integer $n \geq 0$.

We have

$$j|x| + |y| = |x^j y| = |z^\ell| = \ell(j-1)|x| + \ell|u|,$$

and thus $|y| = (\ell j - \ell - j)|x| + \ell|u|$. Since $|y| \le |x|$, $|u| > 0$, $j \ge 2$, and $\ell \ge 2$, it follows that $\ell j - \ell - j = 0$, which implies $j = l = 2$. We therefore have $x^2 y = z^2$ and $x^2 = zv$, hence $vy = z$.

Combining $u = rq$, $u' = qr$, and $v = (rq)^n r$ with $x = vu'$, $z = x^{j-1}u = xu = vu'u$, and $vy = z$, we obtain the solution C with $m = n+1$. The assumption $|y| \le |x|$ implies $m \ge 2$.

Case 3: $j = 1, k \ge 2, y^k \le_s z$.
We have $z = qy^k$ for some word q. Noticing that $x = z^{\ell-1}q$ yields the solution D.

Case 4: $j = 1, k \ge 2, z <_s y^k$.
This case is analogous to the second part of Case 2. Using the Periodicity lemma, we obtain $uy^{k-1} = z$, $y^k = vz$, and $y = vu$ with nonempty u and v. As v is a suffix of z, it is also a suffix of y, and we have $y = vu = u'v$ for some u'. Plugging the solution of the last conjugation equality, namely $u' = rq$, $u = qr$, $v = (rq)^n r$, into $y = u'v$, $z = uy^{k-1}$ and $z^{\ell-1} = xv$ gives the solution E with $m = n+1$.

Finally, the words r and q do not commute since x and y, which are generated by r and q, do not commute.

The proof is completed by a direct verification of the converse. □

We now show that, for a given not primitivity preserving binary code, there is a unique pair of exponents (j,k) such that $x^j y^k$ is imprimitive.

Lemma 1 (LS_unique). *Let $B = \{x,y\}$ be a code. Assume $j, k, j', k' \ge 1$. If both $x^j y^k$ and $x^{j'} y^{k'}$ are imprimitive, then $j = j'$ and $k = k'$.*

Proof. Let z_1, z_2 be primitive words and $\ell, \ell' \ge 2$ be such that

$$x^j y^k = z_1^\ell \quad \text{and} \quad x^{j'} y^{k'} = z_2^{\ell'}. \tag{1}$$

Since B is a code, the words x and y do not commute. We proceed by contradiction.

Case 1: First, assume that $j = j'$ and $k \ne k'$.
Let, without loss of generality, $k < k'$. From (1) we obtain $z_1^\ell y^{k'-k} = z_2^{\ell'}$. The case $k' - k \ge 2$ is impossible due to the Lyndon-Schützenberger theorem. Hence $k' - k = 1$. This is another place where the formalization triggered a simple and nice general lemma (easily provable by the Periodicity lemma) which will turn out to be useful also in the proof of Theorem 4. Namely, the lemma `imprim_ext_suf_comm` claims that if both uv, and uvv are imprimitive, then u and v commute. We apply this lemma to $u = x^j y^{k-1}$ and $v = y$, obtaining a contradiction with the assumption that x and y do not commute.

Case 2. The case $k = k'$ and $j \ne j'$ is symmetric to Case 1.

Case 3. Let finally $j \ne j'$ and $k \ne k'$. The Lyndon-Schützenberger theorem implies that either j or k is one, and similarly either j' or k' is one. We can therefore assume that $k = j' = 1$ and $k', j \geq 2$. Moreover, we can assume that $|y| \leq |x|$. Indeed, in the opposite case, we can consider the words $y^k x^j$ and $y^{k'} x^{j'}$ instead, which are also both imprimitive.

Theorem 2 now allows only the case C for the equality $x^j y = z_1^\ell$. We therefore have $j = \ell = 2$ and $x = (rq)^m r$, $y = qrrq$ for an integer $m \geq 2$ and some noncommuting words r and q. Since $y = qrrq$ is a suffix of z_2^ℓ, this implies that z_2 and rq do not commute. Consider the word $x \cdot qr = (rq)^m rqr$, which is a prefix of xy, and therefore also of z_2^ℓ. This means that $x \cdot qr$ has two periodic roots, namely rq and z_2, and the Periodicity lemma implies that $|x \cdot qr| < |rq| + |z_2|$. Hence x is shorter than z_2. The equality $xy^{k'} = z_2^{\ell'}$, with $\ell' \geq 2$, now implies on one hand that $rqrq$ is a prefix of z_2, and on the other hand that z_2 is a suffix of $y^{k'}$. It follows that $rqrq$ is a factor of $(qrrq)^k$. Hence $rqrq$ and $qrrq$ are conjugate, thus they both have a period of length $|rq|$, which implies $qr = rq$. This is a contradiction. □

The rest of the paper, and therefore also of the proof of Theorem 1, is organized as follows. In Sect. 4, we introduce a general theory of interpretations, which is behind the main idea of the proof, and apply it to the (relatively simple) case of a binary code with words of the same length. In Sect. 5 we characterize the unique disjoint extendable $\{x, y\}$-interpretation of the square of the longer word x. This is a result of independent interest, and also the cornerstone of the proof of Theorem 1 which is completed in Sect. 6 by showing that a word containing at least two x's witnessing that $\{x, y\}$ is not primitivity preserving is conjugate with $[x, x, y]$.

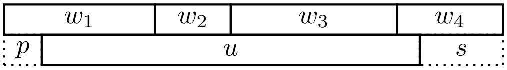
4 Interpretations and the Main Idea

Let X be a code, let u be a factor of $\texttt{concat}\,\mathbf{w}$ for some $\mathbf{w} \in \texttt{lists}\,X$. The natural question is to decide how u can be produced as a factor of words from X, or, in other words, how it can be interpreted in terms of X. This motivates the following definition.

Definition 2. *Let X be a set of words over Σ. We say that the triple $(p, s, \mathbf{w}) \in \Sigma^* \times \Sigma^* \times \texttt{lists}\,X$ is an X-*interpretation *of a word $u \in \Sigma^*$ if*

- $\mathbf{w}$ *is nonempty;*
- $p \cdot u \cdot s = \texttt{concat}\,\mathbf{w}$*;*
- $p <_p \texttt{hd}\,\mathbf{w}$ *and*
- $s <_s \texttt{last}\,\mathbf{w}$*.*

The definition is illustrated by the following figure, where $\mathbf{w} = [w_1, w_2, w_3, w_4]$:



The second condition of the definition motivates the notation $p\,u\,s \sim_{\mathcal{I}} \mathbf{w}$ for the situation when $(p, s, \mathbf{w})$ is an X-interpretation of u.

Remark 1. For sake of historical reference, we remark that our definition of X-interpretation differs from the one used in [1]. Their formulation of the situation depicted by the above figure would be that u is interpreted by the triple $(s', w_2 \cdot w_3, p')$ where $p \cdot s' = w_1$ and $p' \cdot s = w_4$. This is less convenient for two reasons. First, the decomposition of $w_2 \cdot w_3$ into $[w_2, w_3]$ is only implicit here (and even ambiguous if X is not a code). Second, while it is required that the the words p' and s' are a prefix and a suffix, respectively, of an element from X, the identity of that element is left open, and has to be specified separately.

If u is a nonempty element of $\langle X \rangle$ and $u = \texttt{concat}\, \mathbf{u}$ for $\mathbf{u} \in \texttt{lists}\, X$, then the X-interpretation $\varepsilon\, u\, \varepsilon \sim_{\mathcal{I}} \mathbf{u}$ is called *trivial.* Note that the trivial X-interpretation is unique if X is a code.

As nontrivial X-interpretations of elements from $\langle X \rangle$ are of particular interest, the following two concepts are useful.

Definition 3. *An X-interpretation $p\, u\, s \sim_{\mathcal{I}} \mathbf{w}$ of $u = \texttt{concat}\, \mathbf{u}$ is called*

- disjoint *if* $\texttt{concat}\, \mathbf{w}' \neq p \cdot \texttt{concat}\, \mathbf{u}'$ *whenever* $\mathbf{w}' \leq_p \mathbf{w}$ *and* $\mathbf{u}' \leq_p \mathbf{u}$.
- extendable *if* $p \leq_s w_p$ *and* $s \leq_p w_s$ *for some elements* $w_p, w_s \in \langle X \rangle$.

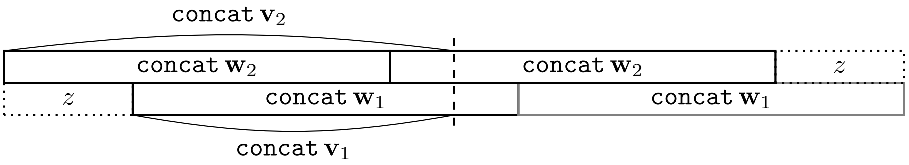
Note that a disjoint X-interpretation is not trivial, and that being disjoint is relative to a chosen factorization $\mathbf{u}$ of u (which is nevertheless unique if X is a code).

The definitions above are naturally motivated by **the main idea** of the characterization of sets X that do not preserve primitivity, which dates back to Lentin and Schützenberger [9]. If $\mathbf{w}$ is primitive, while $\texttt{concat}\, \mathbf{w}$ is imprimitive, say $\texttt{concat}\, \mathbf{w} = z^k$, $k \geq 2$, then the shift by z provides a nontrivial and extendable X-interpretation of $\texttt{concat}\, \mathbf{w}$. (In fact, $k-1$ such nontrivial interpretations). Moreover, the following lemma, formulated in a more general setting of two words $\mathbf{w}_1$ and $\mathbf{w}_2$, implies that the X-interpretation is disjoint if X is a code.

Lemma 2 (shift_interpret, shift_disjoint)**.** *Let X be a code. Let $\mathbf{w}_1, \mathbf{w}_2 \in \texttt{lists}\, X$ be such that $z \cdot \texttt{concat}\, \mathbf{w}_1 = \texttt{concat}\, \mathbf{w}_2 \cdot z$ where $z \notin \langle X \rangle$. Then $z \cdot \texttt{concat}\, \mathbf{v}_1 \neq \texttt{concat}\, \mathbf{v}_2$, whenever $\mathbf{v}_1 \leq_p \mathbf{w}_1^n$ and $\mathbf{v}_2 \leq_p \mathbf{w}_2^n$, $n \in \mathbb{N}$.*

In particular, $\texttt{concat}\, \mathbf{u}$ has a disjoint extendable X-interpretation for any prefix $\mathbf{u}$ of $\mathbf{w}_1$.

The excluded possibility is illustrated by the following figure.



Proof. First, note that $z \cdot \texttt{concat}\,\mathbf{w}_1^n = \texttt{concat}\,\mathbf{w}_2^n \cdot z$ for any n. Let $\mathbf{w}_1^n = \mathbf{v}_1 \cdot \mathbf{v}_1'$ and $\mathbf{w}_2^n = \mathbf{v}_2 \cdot \mathbf{v}_2'$. If $z \cdot \texttt{concat}\,\mathbf{v}_1 = \texttt{concat}\,\mathbf{v}_2$, then also $\texttt{concat}\,\mathbf{v}_2' \cdot z = \texttt{concat}\,\mathbf{v}_1'$. This contradicts $z \notin \langle X \rangle$ by the stability condition.

An extendable X-interpretation of $\mathbf{u}$ is induced by the fact that $\texttt{concat}\,\mathbf{u}$ is covered by $\texttt{concat}(\mathbf{w}_2 \cdot \mathbf{w}_2)$. The interpretation is disjoint by the first part of the proof. □

In order to apply the above lemma to the imprimitive $\texttt{concat}\,\mathbf{w} = z^k$ of a primitive $\mathbf{w}$, set $\mathbf{w}_1 = \mathbf{w}_2 = \mathbf{w}$. The assumption $z \notin \langle X \rangle$ follows from the primitivity of $\mathbf{w}$: indeed, if $z = \texttt{concat}\,\mathbf{z}$, with $\mathbf{z} \in \texttt{lists}\,X$, then $\mathbf{w} = \mathbf{z}^k$ since B is a code.

We first apply the main idea to a relatively simple case of nontrivial $\{x, y\}$-interpretation of the word $x \cdot y$ where x and y are of the same length.

Lemma 3 (uniform_square_interp). *Let $B = \{x, y\}$ be a code with $|x| = |y|$. Let $p\ (x \cdot y)\ s \sim_{\mathcal{I}} \mathbf{v}$ be a nontrivial B-interpretation. Then $\mathbf{v} = [x, y, x]$ or $\mathbf{v} = [y, x, y]$ and $x \cdot y$ is imprimitive.*

Proof. From $p \cdot x \cdot y \cdot s = \texttt{concat}\,\mathbf{v}$, it follows, by a length argument, that $|\mathbf{v}|$ is three. A straightforward way to prove the claim is to consider all eight possible candidates. In each case, it is then a routine few line proof that shows that $x = y$, unless $\mathbf{v} = [x, y, x]$ or $\mathbf{v} = [y, x, y]$, which we omit. In the latter cases, $x \cdot y$ is a nontrivial factor of its square $(x \cdot y) \cdot (x \cdot y)$, which yields the imprimitivity of $x \cdot y$. □

The previous (sketch of the) proof nicely illustrates on a small scale the advantages of formalization. It is not necessary to choose between a tedious elementary proof for sake of completeness on one hand, and the suspicion that something was missed on the other hand (leaving aside that the same suspicion typically remains even after the tedious proof). A bit ironically, the most difficult part of the formalization is to show that $\mathbf{v}$ is indeed of length three, which needs no further justification in a human proof.

We have the following corollary which is a variant of Theorem 4, and also illustrates the main idea of its proof.

Lemma 4 (bin_imprim_not_conjug). *Let $B = \{x, y\}$ be a binary code with $|x| = |y|$. If $\mathbf{w} \in \texttt{lists}\,B$ is such that $|\mathbf{w}| \geq 2$, $\mathbf{w}$ is primitive, and $\texttt{concat}\,\mathbf{w}$ is imprimitive, then x and y are not conjugate.*

Proof. Since $\mathbf{w}$ is primitive and of length at least two, it contains both letters x and y. Therefore, it has either $[x, y]$ or $[y, x]$ as a factor. The imprimitivity of $\texttt{concat}\,\mathbf{w}$ yields a nontrivial B-interpretation of $x \cdot y$, which implies that $x \cdot y$ is not primitive by Lemma 3.

Let x and y be conjugate, and let $x = r \cdot q$ and $y = q \cdot r$. Since $x \cdot y = r \cdot q \cdot q \cdot r$ is imprimitive, also $r \cdot r \cdot q \cdot q$ is imprimitive. Then r and q commute by the theorem of Lyndon and Schützenberger, a contradiction with $x \neq y$. □

5 Binary Interpretation of a Square

Let $B = \{x, y\}$ be a code such that $|y| \leq |x|$. In accordance with the main idea, the core technical component of the proof is the description of the disjoint extendable B-interpretations of the square x^2. This is a very nice result which is relatively simple to state but difficult to prove, and which is valuable on its own. As we mentioned already, it can be obtained from Théorème 2.1 and Lemme 3.1 in [1].

Theorem 3 (`square_interp_ext.sq_ext_interp`)**.** *Let $B = \{x, y\}$ be a code such that $|y| \leq |x|$, both x and y are primitive, and x and y are not conjugate.*

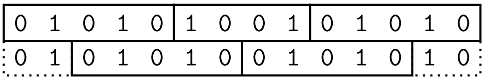
Let $p\,(x \cdot x)\,s \sim_{\mathcal{I}} \mathbf{w}$ be a disjoint extendable B-interpretation. Then

$$\mathbf{w} = [x, y, x], \qquad s \cdot p = y, \qquad p \cdot x = x \cdot s.$$

In order to appreciate the theorem, note that the definition of interpretation implies

$$p \cdot x \cdot x \cdot s = x \cdot y \cdot x,$$

hence $x \cdot y \cdot x = (p \cdot x)^2$. This will turn out to be the only way how primitivity may not be preserved if x occurs at least twice in $\mathbf{w}$. Here is an example with $x = \texttt{01010}$ and $y = \texttt{1001}$:



Proof. By the definition of a disjoint interpretation, we have $p \cdot x \cdot x \cdot s = \texttt{concat}\,\mathbf{w}$, where $p \neq \varepsilon$ and $s \neq \varepsilon$. A length argument implies that $\mathbf{w}$ has length at least three. Since a primitive word is not a nontrivial factor of its square, we have $\mathbf{w} = [\texttt{hd}\,\mathbf{w}] \cdot [y]^k \cdot [\texttt{last}\,\mathbf{w}]$, with $k \geq 1$. Since the interpretation is disjoint, we can split the equality into $p \cdot x = \texttt{hd}\,\mathbf{w} \cdot y^m \cdot u$ and $x \cdot s = v \cdot y^\ell \cdot \texttt{last}\,\mathbf{w}$, where $y = u \cdot v$, both u and v are nonempty, and $k = \ell + m + 1$. We want to show $\texttt{hd}\,\mathbf{w} = \texttt{last}\,\mathbf{w} = x$ and $m = \ell = 0$. The situation is mirror symmetric so we can solve cases two at a time.

If $\texttt{hd}\,\mathbf{w} = \texttt{last}\,\mathbf{w} = y$, then powers of x and y share a factor of length at least $|x| + |y|$. Since they are primitive, this implies that they are conjugate, a contradiction. The same argument applies when $\ell \geq 1$ and $\texttt{hd}\,\mathbf{w} = y$ (if $m \geq 1$ and $\texttt{last}\,\mathbf{w} = y$ respectively). Therefore, in order to prove $\texttt{hd}\,\mathbf{w} = \texttt{last}\,\mathbf{w} = x$, it remains to exclude the case $\texttt{hd}\,\mathbf{w} = y$, $\ell = 0$ and $\texttt{last}\,\mathbf{w} = x$ ($\texttt{last}\,\mathbf{w} = y$, $m = 0$ and $\texttt{hd}\,\mathbf{w} = x$ respectively). This is covered by one of the technical lemmas that we single out:

Lemma 5 (`pref_suf_pers_short`)**.** *Let $x \leq_p v \cdot x$, $x \leq_s p \cdot u \cdot v \cdot u$ and $|x| > |v \cdot u|$ with $p \in \langle\{u, v\}\rangle$. Then $u \cdot v = v \cdot u$.*

This lemma indeed excludes the case we wanted to exclude, since the conclusion implies that y is not primitive. We skip the proof of the lemma here and make instead an informal comment. Note that v is a period root of x. In other words, x is a factor of v^{ω}. Therefore, with the stronger assumption that $v \cdot u \cdot v$ is a factor of x, the conclusion follows easily by the familiar principle that v being a factor of v^{ω} "synchronizes" primitive roots of v. Lemma 5 then exemplifies one of the virtues of formalization, which makes it easy to generalize auxiliary lemmas, often just by following the most natural proof and checking its minimal necessary assumptions.

Now we have $\texttt{hd}\,\mathbf{w} = \texttt{last}\,\mathbf{w} = x$, hence $p \cdot x = x \cdot y^m \cdot u$ and $x \cdot s = v \cdot y^{\ell} \cdot x$. The natural way to describe this scenario is to observe that x has both the (prefix) period root $v \cdot y^{\ell}$, and the suffix period root $y^m \cdot u$. Using again Lemma 5, we exclude situations when $\ell = 0$ and $m \geq 1$ ($m = 0$ and $\ell \geq 1$ resp.). It therefore remains to deal with the case when both m and ℓ are positive. We divide this into four lemmas according to the size of the overlap the prefix $v \cdot y^{\ell}$ and the suffix $y^m \cdot u$ have in x. More exactly, the cases are:

- $\left|v \cdot y^{\ell}\right| + |y^m \cdot u| \leq |x|$
- $|x| < \left|v \cdot y^{\ell}\right| + |y^m \cdot u| \leq |x| + |u|$
- $|x| + |u| < \left|v \cdot y^{\ell}\right| + |y^m \cdot u| < |x| + |u \cdot v|$
- $|x| + |u \cdot v| \leq \left|v \cdot y^{\ell}\right| + |y^m \cdot u|$

and they are solved by an auxiliary lemma each. The first three cases yield that u and v commute, the first one being a straightforward application of the Periodicity lemma. The last one is also straightforward application of the "synchronization" idea. It implies that $x \cdot x$ is a factor of y^{ω}, a contradiction with the assumption that x and y are primitive and not conjugate. Consequently, the technical, tedious part of the whole proof is concentrated in lemmas dealing with the second, and the third case (see lemmas `short_overlap` and `medium_overlap` in the theory `Binary_Square_Interpretation.thy`). The corresponding proofs are further analyzed and decomposed into more elementary claims in the formalization, where further details can be found.

This completes the proof of $\mathbf{w} = [x, y, x]$. A byproduct of the proof is the description of words x, y, p and s. Namely, there are non-commuting words r and t, and integers m, k and ℓ such that

$$x = (rt)^{m+1} \cdot r, \qquad y = (tr)^{k+1} \cdot (rt)^{\ell+1}, \qquad p = (rt)^{k+1}, \qquad s = (tr)^{\ell+1}\,.$$

The second claim of the present theorem, that is, $y = s \cdot p$ is then equivalent to $k = \ell$, and it is an easy consequence of the assumption that the interpretation is extendable. □

6 The Witness with Two x's

In this section, we characterize words witnessing that $\{x, y\}$ is not primitivity preserving and containing at least two x's.

Theorem 4 **(**`bin_imprim_longer_twice`**).** *Let $B = \{x, y\}$ be a code such that $|y| \leq |x|$. Let $\mathbf{w} \in$ `lists` $\{x, y\}$ be a primitive word which contains x at least twice such that* `concat` $\mathbf{w}$ *is imprimitive.*

Then $\mathbf{w} \sim [x, x, y]$ and both x and y are primitive.

We divide the proof in three steps.

The Core Case. We first prove the claim with two additional assumptions which will be subsequently removed. Namely, the following lemma shows how the knowledge about the B-interpretation of $x \cdot x$ from the previous section is used. The additional assumptions are displayed as items.

Lemma 6 **(**`bin_imprim_primitive`**).** *Let $B = \{x, y\}$ be a code with $|y| \leq |x|$ where*

- *both x and y are primitive,*

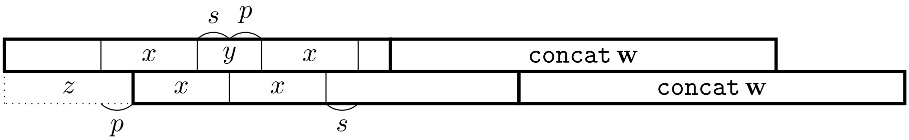
and let $\mathbf{w} \in$ `lists` B be primitive such that `concat` $\mathbf{w}$ *is imprimitive, and*

- *$[x, x]$ is a cyclic factor of $\mathbf{w}$.*

Then $\mathbf{w} \sim [x, x, y]$.

Proof. Choosing a suitable conjugate of $\mathbf{w}$, we can suppose, without loss of generality, that $[x, x]$ is a prefix of $\mathbf{w}$. Now, we want to show $\mathbf{w} = [x, x, y]$. Proceed by contradiction and assume $\mathbf{w} \neq [x, x, y]$. Since $\mathbf{w}$ is primitive, this implies $\mathbf{w} \cdot [x, x, y] \neq [x, x, y] \cdot \mathbf{w}$.

By Lemma 4, we know that x and y are not conjugate. Let `concat` $\mathbf{w} = z^k$, $2 \leq k$ and z primitive. Lemma 2 yields a disjoint extendable B-interpretation of $(\texttt{concat}\,\mathbf{w})^2$. In particular, the induced disjoint extendable B-interpretation of the prefix $x \cdot x$ is of the form $p\,(x \cdot x)\,s \sim_{\mathcal{I}} [x, y, x]$ by Theorem 3:



Let $\mathbf{p}$ be the prefix of $\mathbf{w}$ such that `concat` $\mathbf{p} \cdot p = z$. Then

$$\texttt{concat}(\mathbf{p} \cdot [x, y]) = z \cdot (x \cdot p), \quad \texttt{concat}\,[x, x, y] = (x \cdot p)^2, \quad \texttt{concat}\,\mathbf{w} = z^k,$$

and we want to show $z = xp$, which will imply $\texttt{concat}([x, x, y] \cdot \mathbf{w}) = \texttt{concat}(\mathbf{w} \cdot [x, x, y])$, hence $\mathbf{w} = [x, x, y]$ since $\{x, y\}$ is a code, and both $\mathbf{w}$ and $[x, x, y]$ are primitive.

Again, proceed by contradiction, and assume $z \neq xp$. Then, since both z and $x \cdot p$ are primitive, they do not commute. We now have two binary codes, namely $\{\mathbf{w}, [x, x, y]\}$ and $\{z, xp\}$. The following two equalities, (2) and (3) exploit the fundamental property of longest common prefixes of elements of binary codes mentioned in Sect. 2. In particular, we need the following lemma:

Lemma 7 (`bin_code_lcp_concat`). *Let $X = \{u_0, u_1\}$ be a binary code, and let $\mathbf{z}_0, \mathbf{z}_1 \in$ `lists` X be such that* `concat` $\mathbf{z}_0$ *and* `concat` $\mathbf{z}_1$ *are not prefix-comparable. Then*

$$(\texttt{concat}\,\mathbf{z}_0) \wedge_p (\texttt{concat}\,\mathbf{z}_1) = \texttt{concat}(\mathbf{z}_0 \wedge_p \mathbf{z}_1) \cdot (u_0 \wedge u_1).$$

See Sect. 8 for more comments on this property. Denote $\alpha_{z,xp} = z \cdot xp \wedge_p xp \cdot z$. Then also $\alpha_{z,xp} = z^k \cdot (xp)^2 \wedge_p (xp)^2 \cdot z^k$. Similarly, let $\alpha_{x,y} = x \cdot y \wedge_p y \cdot x$. Then Lemma 7 yields

$$\begin{aligned} \alpha_{z,xp} &= \texttt{concat}(\mathbf{w} \cdot [x,x,y]) \wedge_p \texttt{concat}([x,x,y] \cdot \mathbf{w}) \\ &= \texttt{concat}(\mathbf{w} \cdot [x,x,y] \wedge_p [x,x,y] \cdot \mathbf{w}) \cdot \alpha_{x,y} \end{aligned} \tag{2}$$

and also

$$\begin{aligned} z \cdot \alpha_{z,xp} =& \texttt{concat}(\mathbf{w} \cdot \mathbf{p} \cdot [x,y]) \wedge_p \texttt{concat}(\mathbf{p} \cdot [x,y] \cdot \mathbf{w}) \\ =& \texttt{concat}(\mathbf{w} \cdot \mathbf{p} \cdot [x,y] \wedge_p \mathbf{p} \cdot [x,y] \cdot \mathbf{w}) \cdot \alpha_{x,y}. \end{aligned} \tag{3}$$

Denote

$$\mathbf{v}_1 = \mathbf{w} \cdot [x,x,y] \wedge_p [x,x,y] \cdot \mathbf{w}, \qquad \mathbf{v}_2 = \mathbf{w} \cdot \mathbf{p} \cdot [x,y] \wedge_p \mathbf{p} \cdot [x,y] \cdot \mathbf{w}.$$

From (2) and (3) we now have $z \cdot \texttt{concat}\,\mathbf{v}_1 = \texttt{concat}\,\mathbf{v}_2$. Since $\mathbf{v}_1$ and $\mathbf{v}_2$ are prefixes of some $\mathbf{w}^n$, we have a contradiction with Lemma 2. □

Dropping the Primitivity Assumption. We first deal with the situation when x and y are not primitive. A natural idea is to consider the primitive roots of x and y instead of x and y. This means that we replace the word $\mathbf{w}$ with $\mathcal{R}\mathbf{w}$, where $\mathcal{R}$ is the morphism mapping $[x]$ to $[\rho\, x]^{e_x}$ and $[y]$ to $[\rho\, y]^{e_y}$ where $x = (\rho\, x)^{e_x}$ and $y = (\rho\, y)^{e_y}$. For example, if $x = abab$ and $y = aa$, and $\mathbf{w} = [x, y, x] = [abab, aa, abab]$, then $\mathcal{R}\mathbf{w} = [ab, ab, a, a, ab, ab]$.

Let us check which hypotheses of Lemma 6 are satisfied in the new setting, that is, for the code $\{\rho\, x, \rho\, y\}$ and the word $\mathcal{R}\mathbf{w}$. The following facts are not difficult to see.

- `concat` $\mathbf{w} =$ `concat`$(\mathcal{R}\mathbf{w})$;
- if $[c, c]$, $c \in \{x, y\}$, is a cyclic factor $\mathbf{w}$, then $[\rho\, c, \rho\, c]$ is a cyclic factor of $\mathcal{R}\mathbf{w}$.

The next required property:

- if $\mathbf{w}$ is primitive, then $\mathcal{R}\mathbf{w}$ is primitive;

deserves more attention. It triggered another little theory of our formalization which can be found in locale `sings_code`. Note that it fits well into our context, since the claim is that $\mathcal{R}$ is a primitivity preserving morphism, which implies that its image on the singletons $[x]$ and $[y]$ forms a primitivity preserving set of words, see theorem `code.roots_prim_morph`.

Consequently, the only missing hypothesis preventing the use of Lemma 6 is $|y| \leq |x|$ since it may happen that $|\rho\, x| < |\rho\, y|$. In order to solve this difficulty, we shall ignore for a while the length difference between x and y, and obtain the following intermediate lemma.

Lemma 8 (`bin_imprim_both_squares`, `bin_imprim_both_squares_prim`). *Let $B = \{x, y\}$ be a code, and let $\mathbf{w} \in$ `lists` B be a primitive word such that* `concat` $\mathbf{w}$ *is imprimitive. Then* $\mathbf{w}$ *cannot contain both* $[x, x]$ *and* $[y, y]$ *as cyclic factors.*

Proof. Assume that $\mathbf{w}$ contains both $[x, x]$ and $[y, y]$ as cyclic factors.

Consider the word $\mathcal{R}\mathbf{w}$ and the code $\{\rho\, x, \rho\, y\}$. Since $\mathcal{R}\mathbf{w}$ contains both $[\rho\, x, \rho\, x]$ and $[\rho\, y, \rho\, y]$, Lemma 6 implies that $\mathcal{R}\mathbf{w}$ is conjugate either with the word $[\rho\, x, \rho\, x, \rho\, y]$ or with $[\rho\, y, \rho\, y, \rho\, x]$, which is a contradiction with the assumed presence of both squares. □

Concluding the Proof by Gluing. It remains to deal with the existence of squares. We use an idea that is our main innovation with respect to the proof from [1], and contributes significantly to the reduction of length of the proof, and hopefully also to its increased clarity. Let $\mathbf{w}$ be a list over a set of words X. The idea is to choose one of the words, say $u \in X$, and to concatenate (or "glue") blocks of u's to words following them. For example, if $\mathbf{w} = [u, v, u, u, z, u, z]$, then the resulting list is $[uv, uuz, uz]$. This procedure is in the general case well defined on lists whose last "letter" is not the chosen one and it leads to a new alphabet $\{u^i \cdot v \mid v \neq u\}$ which is a code if and only if X is. This idea is used in an elegant proof of the Graph lemma (see [8] and [2]). In the binary case, which is of interest here, if $\mathbf{w}$ in addition does not contain a square of a letter, say $[x, x]$, then the new code $\{x \cdot y, y\}$ is again binary. Moreover, the resulting glued list $\mathbf{w}'$ has the same concatenation, and it is primitive if (and only if) $\mathbf{w}$ is. Note that gluing is in this case closely related to the Nielsen transformation $y \mapsto x^{-1}y$ known from the theory of automorphisms of free groups.

Induction on $|\mathbf{w}|$ now easily leads to the proof of Theorem 4.

Proof (of Theorem 4). If $\mathbf{w}$ contains y at most once, then we are left with the equation $x^j \cdot y = z^\ell$, $\ell \geq 2$. The equality $j = 2$ follows from the Periodicity lemma, see Case 2 in the proof of Theorem 2.

Assume for contradiction that y occurs at least twice in $\mathbf{w}$. Lemma 8 implies that at least one square, $[x, x]$ or $[y, y]$ is missing as a cyclic factor. Let $\{x', y'\} = \{x, y\}$ be such that $[x', x']$ is not a cyclic factor of $\mathbf{w}$. We can therefore perform the gluing operation, and obtain a new, strictly shorter word $\mathbf{w}' \in$ `lists` $\{x' \cdot y', y'\}$. The longer element $x' \cdot y'$ occurs at least twice in $\mathbf{w}'$, since the number of its occurrences in $\mathbf{w}'$ is the same as the number of occurrences of x' in $\mathbf{w}$, the latter word containing both letters at least twice by assumption. Moreover, $\mathbf{w}'$ is primitive, and `concat` $\mathbf{w}' =$ `concat` $\mathbf{w}$ is imprimitive. Therefore, by induction on $|\mathbf{w}|$, we have $\mathbf{w}' \sim [x' \cdot y', x' \cdot y', y']$. In order to show that this is not possible we can successfully reuse the lemma `imprim_ext_suf_comm` mentioned in the proof of Lemma 1, this time for $u = x'y'x'$ and $v = y'$. The words u and v do not commute because x' and y' do not commute. Since uv is imprimitive, the word $uvv \sim$ `concat` $\mathbf{w}'$ is primitive. □

This also completes the proof of our main target, Theorem 1.

7 Additional Notes on the Formalization

The formalization is a part of an evolving combinatorics on words formalization project. It relies on its backbone session, called CoW, a version of which is also available in the Archive of Formal Proofs [15]. This session covers basics concepts in combinatorics on words including the Periodicity lemma. An overview is available in [8].

The evolution of the parent session CoW continued along with the presented results and its latest stable version is available at our repository [16]. The main results are part of another Isabelle session CoW_Equations, which, as the name suggests, aims at dealing with word equations. We have greatly expanded its elementary theory `Equations_Basic.thy` which provides auxiliary lemmas and definitions related to word equations. Noticeably, it contains the definition `factor_interpretation` (Definition 2) and related facts.

Two dedicated theories were created: `Binary_Square_Interpretation.thy` and `Binary_Code_Imprimitive.thy`. The first contains lemmas and locales dealing with $\{x, y\}$-interpretation of the square xx (for $|y| \leq |x|$), culminating in Theorem 3. The latter contains Theorems 1 and 4.

Another outcome was an expansion of formalized results related to the Lyndon-Schützenberger theorem. This result, along with many useful corollaries, was already part of the backbone session CoW, and it was newly supplemented with the parametric solution of the equation $x^j y^k = z^\ell$, specifically Theorem 2 and Lemma 1. This formalization is now part of CoW_Equations in the theory `Lyndon_Schutzenberger.thy`.

Similarly, the formalization of the main results triggered a substantial expansion of existing support for the idea of gluing as mentioned in Sect. 6. Its reworked version is now in a separate theory called `Glued_Codes.thy` (which is part of the session CoW_Graph_Lemma).

Let us give a few concrete highlights of the formalization. A very useful tool, which is part of the CoW session, is the `reversed` attribute. The attribute produces a symmetrical fact where the symmetry is induced by the mapping **rev**, i.e., the mapping which reverses the order of elements in a list. For instance, the fact stating that if p is a prefix of v, then p a prefix of $v \cdot w$, is transformed by the reversed attribute into the fact saying that if s is suffix of v, then s is a suffix of $w \cdot v$. The attribute relies on ad hoc defined rules which induce the symmetry. In the example, the main reversal rule is

$$(\mathrm{rev}\ \mathrm{u} \leq \mathrm{p}\ \ \mathrm{rev}\ \ \mathrm{v}) = \mathrm{u} \leq \mathrm{s}\ \ \mathrm{v}.$$

The attribute is used frequently in the present formalization. For instance, Fig. 1 shows the formalization of the proof of Cases 1 and 2 of Theorem 1. Namely, the proof of Case 2 is smoothly deduced from the lemma that deals with Case 1, avoiding writing down the same proof again up to symmetry. See [13] for more details on the symmetry and the attribute `reversed`.

To be able to use this attribute fully in the formalization of main results, it needed to be extended to be able to deal with elements of type `'a list list`, as the constant `factor_interpretation` is of the function type over this exact

```
proof(cases)
  case 1
  then show ?thesis
    using LS-unique-same
      assms(1, 4−8) by blast
next
  case 2
  then show ?thesis
    using LS-unique-same[reversed]
      assms(1, 3, 5−8) by blast
```

(a) Using the `reversed` attribute to solve symmetric cases.

```
  have primitive [x,x,y]
    using ⟨x ≠ y⟩
   by primitivity-inspection
```

```
  from ⟨|ws| = 3⟩ ⟨ws ∈ lists {x,y}⟩
⟨x ≠ y⟩ ⟨[x, x] ≤f ws · ws⟩
⟨[y, y] ≤f ws · ws⟩
  show False
   by list-inspection simp-all
```

```
  from ⟨p · t · s = t · t · p⟩
  have p · t = t · p
    by mismatch
```

(b) Methods `primitivity_inspection`, `list_inspection` and `mismatch`.

Fig. 1. Highlights from the formalization in Isabelle/HOL.

type. The new theories of the session CoW_Equations contain almost 50 uses of this attribute.

The second highlight of the formalization is the use of simple but useful proof methods. The first method, called `primitivity_inspection`, is able to show primitivity or imprimitivity of a given word.

Another method named `list_inspection` is used to deal with claims that consist of straightforward verification of some property for a set of words given by their length and alphabet. For instance, this method painlessly concludes the proof of lemma `bin_imprim_both_squares_prim`. The method divides the goal into eight easy subgoals corresponding to eight possible words. All goals are then discharged by `simp_all`.

The last method we want to mention is `mismatch`. It is designed to prove that two words commute using the property of a binary code mentioned in Sect. 2 and explained in Sect. 8. Namely, if a product of words from $\{x, y\}$ starting with x shares a prefix of length at least $|xy|$ with another product of words from $\{x, y\}$, this time starting with y, then x and y commute. Examples of usage of the attribute `reversed` and all three methods are given in Fig. 1.

8 Appendix: Background Results in Combinatorics on Words

A periodic root r of w need not be primitive, but it is always possible to consider the corresponding primitive root $\rho\, r$, which is also a periodic root of w. Note that any word has infinitely many periodic roots since we allow r to be longer than w. Nevertheless, a word can have more than one period even if we consider only periods shorter than $|w|$. Such a possibility is controlled by the Periodicity lemma, often called the Theorem of Fine and Wilf (see [6]):

Lemma 9 (`per_lemma_comm`). *If w has a period u and v, i.e., $w \leq_p uw$ and $w \leq_p vw$, with $|u| + |v| - \gcd(|u|, |v|) \leq |w|$, then $uv = vu$.*

Usually, the weaker test $|u| + |v| \leq |w|$ is sufficient to indicate that u and v commute.

Conjugation $u \sim v$ is characterized as follows:

Lemma 10 (`conjugation`). *If $uz = zv$ for nonempty u, then there exists words r and q and an integer k such that $u = rq$, $v = qr$ and $z = (rq)^k r$.*

We have said that w has a periodic root r if it is a prefix of r^ω. If w is a factor, not necessarily a prefix, of r^ω, then it has a periodic root which is a conjugate of r. In particular, if $|u| = |v|$, then $u \sim v$ is equivalent to u and v being mutually factors of a power of the other word.

Commutation of two words is characterized as follows:

Lemma 11 (`comm`). *$xy = yx$ if and only if $x = t^k$ and $y = t^m$ for some word t and some integers $k, m \geq 0$.*

Since every nonempty word has a (unique) primitive root, the word t can be chosen primitive (k or m can be chosen 0 if x or y is empty).

We often use the following theorem, called "the theorem of Lyndon and Schützenberger":

Theorem 5 (`Lyndon_Schutzenberger`). *If $x^j y^k = z^\ell$ with $j \geq 2$, $k \geq 2$ and $\ell \geq 2$, then the words x, y and z commute.*

A crucial property of a primitive word t is that it cannot be a nontrivial factor of its own square. For a general word u, the equality $u \cdot u = p \cdot u \cdot s$ with nonempty p and s implies that all three words p, s, u commute, that is, have a common primitive root t. This can be seen by writing $u = t^k$, and noticing that the presence of a nontrivial factor u inside uu can be obtained exclusively by a shift by several t's. This idea is often described as "synchronization".

Let x and y be two words that do not commute. The longest common prefix of xy and yx is denoted α. Let c_x and c_y be the letter following α in xy and yx respectively. A crucial property of α is that it is a prefix of any sufficiently long word in $\langle\{x, y\}\rangle$. Moreover, if $\mathbf{w} = [u_1, u_2, \ldots, u_n] \in \texttt{lists}\,\{x, y\}$ is such that $\texttt{concat}\,\mathbf{w}$ is longer than α, then $\alpha \cdot [c_x]$ is a prefix of $\texttt{concat}\,\mathbf{w}$ if $u_1 = x$ and $\alpha \cdot [c_y]$ is a prefix of $\texttt{concat}\,\mathbf{w}$ if $u_1 = y$. That is why the length of α is sometimes called "the decoding delay" of the binary code $\{x, y\}$. Note that the property indeed in particular implies that $\{x, y\}$ is a code, that is, it does not satisfy any nontrivial relation. It is also behind our method `mismatch`. Finally, using this property, the proof of Lemma 7 is straightforward.

Acknowledgments. The authors acknowledge support by the Czech Science Foundation grant GAČR 20-20621S.

References

1. Barbin-Le Rest, E., Le Rest, M.: Sur la combinatoire des codes à deux mots. Theor. Comput. Sci. **41**, 61–80 (1985)
2. Berstel, J., Perrin, D., Perrot, J.F., Restivo, A.: Sur le théorème du défaut. J. Algebra **60**(1), 169–180 (1979). http://www.sciencedirect.com/science/article/pii/0021869379901133. https://doi.org/10.1016/0021-8693(79)90113-3
3. Berstel, J., Perrin, D., Reutenauer, C.: Codes and Automata. Cambridge (2010). https://www.ebook.de/de/product/8629820/jean_berstel_dominique_perrin_christophe_reutenauer_codes_and_automata.html
4. Budkina, L.G., Markov, Al.A.: F-semigroups with three generators. Mat. Zametki **14**, 267–277 (1973). Translated from Mat. Zametki **14**(2), 267–277 (1973)
5. Crochemore, M.: Sharp characterizations of squarefree morphisms. Theor. Comput. Sci. **18**(2), 221–226 (1982). https://doi.org/10.1016/0304-3975(82)90023-8
6. Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. Proc. Am. Math. Soc. **16**(1), 109–109 (1965). https://doi.org/10.1090/S0002-9939-1965-0174934-9
7. Harju, T., Nowotka, D.: On the independence of equations in three variables. Theoret. Comput. Sci. **307**(1), 139–172 (2003). https://doi-org.ezproxy.is.cuni.cz/10.1016/S0304-3975(03)00098-7. https://doi.org/10.1016/S0304-3975(03)00098-7
8. Holub, Š., Starosta, Š.: Formalization of basic combinatorics on words. In: Cohen, L., Kaliszyk, C. (eds.) 12th International Conference on Interactive Theorem Proving (ITP 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 193, pp. 22:1–22:17, Dagstuhl, Germany. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://drops.dagstuhl.de/opus/volltexte/2021/13917. https://doi.org/10.4230/LIPIcs.ITP.2021.22
9. Lentin, A., Schützenberger, M.-P.: A combinatorial problem in the theory of free monoids. In: Combinatorial Mathematics and its Applications (Proc. Conf., Univ. North Carolina, Chapel Hill, N.C., 1967), pp. 128–144. University North Carolina Press, Chapel Hill (1969)
10. Lyndon, R.C., Schützenberger, M.-P.: The equation $a^m = b^n c^p$ in a free group. Michigan Math. J. **9**(4), 289–298 (1962). https://doi.org/10.1307/mmj/1028998766
11. Manuch, J.: Defect effect of bi-infinite words in the two-element case. Discret. Math. Theor. Comput. Sci. **4**(2), 273–290 (2001). http://dmtcs.episciences.org/279
12. Mitrana, V.: Primitive morphisms. Inform. Process. Lett. **64**(6), 277–281 (1997). https://doi.org/10.1016/s0020-0190(97)00178-6
13. Raška, M., Starosta, Š.: Producing symmetrical facts for lists induced by the list reversal mapping in Isabelle/HOL (2021). https://arxiv.org/abs/2104.11622
14. Spehner, J.-P.: Quelques problèmes d'extension, de conjugaison et de presentation des sous-monoïdes d'un monoïde libre. Ph.D. thesis, Université Paris VII, Paris (1976)
15. Holub, Š., Raška, M., Starosta, Š.: Combinatorics on words basics. Archive of Formal Proofs, May 2021. https://isa-afp.org/entries/Combinatorics_Words.html. Formal proof development
16. Holub, Š., Raška, M., Starosta, Š.: Combinatorics on words formalized (release v1.6) (2022). https://gitlab.com/formalcow/combinatorics-on-words-formalized

Formula Simplification via Invariance Detection by Algebraically Indexed Types

Takuya Matsuzaki(✉) and Tomohiro Fujita

Tokyo University of Science, 1-3 Kagurazaka, Shinjuku-ku, Tokyo 162-8601, Japan
matuzaki@rs.tus.ac.jp, 1418097@ed.tus.ac.jp

Abstract. We describe a system that detects an invariance in a logical formula expressing a math problem and simplifies it by eliminating variables utilizing the invariance. Pre-defined function and predicate symbols in the problem representation language are associated with algebraically indexed types, which signify their invariance property. A Hindley-Milner style type reconstruction algorithm is derived for detecting the invariance of a problem. In the experiment, the invariance-based formula simplification significantly enhanced the performance of a problem solver based on quantifier-elimination for real-closed fields, especially on the problems taken from the International Mathematical Olympiads.

1 Introduction

It is very common to find an argument marked by the phrase "without loss of generality" (w.l.o.g.) in a mathematical proof by human. An argument of this kind is most often based on the symmetry or the invariance in the problem [9].

Suppose that we are going to prove, by an algebraic method, that the three median lines of a triangle meet at a point (Fig. 1). Six real variables are needed to represent three points on a plane. Since the concepts of 'median lines' and 'meeting at a point' are translation-invariant, we may fix one of the corners at the origin. Furthermore, because these concepts are also invariant under any invertible linear map, we may fix the other two points to, e.g., $(1, 0)$ and $(0, 1)$. Thus, all six variables were eliminated and the task of proof became much easier.

W.l.o.g. arguments may thus have strong impact on the efficiency of inference. It has drawn attention in several research areas including the relative strength of proof systems (e.g., [2,3,12,20]), propositional SAT (e.g., [1,6,8,17,19]), proof assistants [9], and algebraic methods for geometry problem solving [7,10].

Among others, Iwane and Anai [10] share exactly the same objective with us; both aim at solving geometry problems stated in natural language, using an algebraic method as the backend. Logical formulas resulted from mechanical translation of problem text tend to be huge and very redundant, while the computational cost of algebraic methods is generally quite sensitive to the size of the input measured by, e.g., the number of variables. Simplification of the input formula is hence a mandatory part of such a problem-solving system.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 388–406, 2022.
https://doi.org/10.1007/978-3-031-10769-6_24

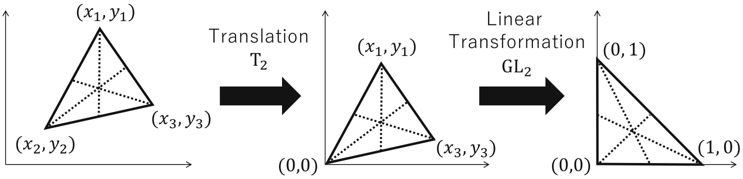


Fig. 1. Variable Elimination w.l.o.g. by Invariance

Iwane and Anai's method operates on the first-order formula of real-closed fields (RCFs), i.e., a quantified boolean combination of equalities and inequalities between polynomials. They proposed to detect the invariance of a problem by testing the invariance of the polynomials under translation, scaling, and rotation. While being conceptually simple, it amounts to discover the geometric property of the problem solely by its algebraic representation. The detection of rotational invariance is especially problematic because, to test that on a system of polynomials, one needs to identify all the pairs (or triples) of variables that originate from the x and y (and z) coordinates of the same points. Thus their algorithm for 2D rotational invariance already incurs a search among a large number of possibilities and they left the detection of 3D rotational invariance untouched. Davenport [7] also suggests essentially the same method.

In this paper, we propose to detect the invariance in a more high-level language than that of RCF. We use algebraically indexed types (AITs) proposed by Atkey et al. [4] as the representation language. In AIT, each symbol in a formula has a type with indices. An indexed-type of a function indicates that its output undergoes the same or a related transformation as the input. The invariances of the functions are combined via type reconstruction and an invariance in a problem is detected.

The contribution of the current paper is summarized as follows:

1. A type reconstruction algorithm for AIT is derived. Atkey et al. [4] laid out the formalism of AIT but did not provide a type inference/reconstruction algorithm. We devised, for a version of AIT, a type reconstruction algorithm that is based on semantic unification in the theory of transformation groups.
2. A set of variable elimination rules are worked out. Type reconstruction in AIT discerns a more fine-grained notion of invariance than previous approaches. We derived a set of elimination rules that covers all cases.
3. The practicality of the proposed method is verified; it significantly enhanced the performance of a problem solver based on quantifier elimination for RCF, especially on the problems from International Mathematical Olympiads.

In the rest of the paper, we first introduce a math problem solver, on which the proposed method was implemented, and summarize the formalism of AIT. We then detail the type reconstruction procedure and the variable elimination rules. We finally present the experimental results and conclude the paper.

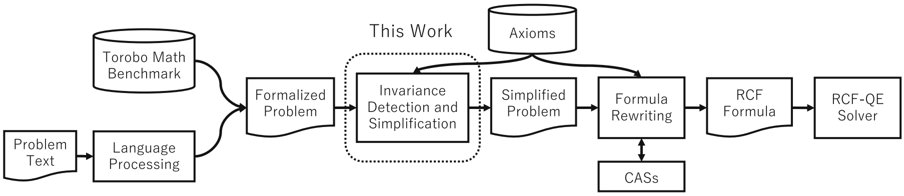


Fig. 2. Overview of Todai Robot Math Problem Solver

```
(Show (forall (A B C D X K L M )
        (-> (&& (is-triangle A B C)
                (= (rad-of-angle (angle B C A)) (* 90 (Degree)))
                (= D (foot-of-perp-line-from-to C (line A B)))
                (on X (seg C D)) (! (= X C)) (! (= X D))
                (on K (seg A X))
                (= (length-of (seg B K)) (length-of (seg B C)))
                (on L (seg B X))
                (= (length-of (seg A L)) (length-of (seg A C)))
                (intersect (seg A L) (seg B K) M))
            (= (length-of (seg M L)) (length-of (seg M K))))))
```

Fig. 3. Example of Manually Formalized Problem (IMO 2012, Problem 5)

2 Todai Robot Math Solver and Problem Library

This work is a part of the development of the Todai Robot Math Problem Solver (henceforth ToroboMath) [13–16]. Figure 2 presents an overview of the system. ToroboMath is targeted at solving pre-university math problems. Our long-term goal is to develop a system that solves problems stated in natural language.

The natural language processing (NLP) module of the system accepts a problem text and derives its logical representation through syntactic analysis. Currently, it produces a correct logical form for around 50% of sentences [13], which is not high enough to cover a wide variety of problems. Although the motivation behind the current work is to cope with the huge formulas produced by the NLP module, we instead used a library of *manually* formalized problems for the evaluation of the formula simplification procedure.

The problem library has been developed along with the ToroboMath system. It contains approximately one thousand math problems collected from several sources including the International Mathematical Olympiads (IMOs). Figure 3 presents a problem that was taken from IMO 2012.

The problems in the library are manually encoded in a polymorphic higher-order language, which is the same language as the output of the NLP module. Table 1 lists some of its primitive types. The language includes a large set of predicate and function symbols that are tailored for formalizing pre-university math problems. Currently, 1387 symbols are defined using 2808 axioms. Figure 4 provides an example of the axioms that defines the predicate `maximum`.

Table 1. Example of Primitive Types

truth values	`Bool`
numbers	`Z` (integers), `Q` (rationals), `R` (reals), `C` (complex)
vectors	`2d.Vec`, `3d.Vec`
geometric objects	`2d.Shape`, `3d.Shape`
angles	`2d.Angle`, `3d.Angle`
sets and lists	`SetOf`(α), `ListOf`(α)

```
(axiom def_maximum
  (forall (set max)
    (<-> (maximum set max)
         (& (elem max set)
            (forall (v)
              (-> (elem v set)
                  (<= v max)))))))
```

Fig. 4. Example of Axiom

The problem solving module of the ToroboMath accepts a formalized problem and iteratively rewrites it using: (1) basic transformations such as $\forall x.(x = \alpha \rightarrow \phi(x)) \Leftrightarrow \phi(\alpha)$ and beta-reduction, (2) simplification of expressions such as polynomial division and integration by computer algebra systems (CASs), and (3) the axioms that define the predicate and function symbols.

Once the rewritten formula is in the language of real-closed fields (RCFs) or Peano arithmetic, it is handed to a solver for the theory. For RCF formulas, we use an implementation of the quantifier-elimination (QE) procedure for RCF based on cylindrical algebraic decomposition. Finally, we solve the resulting quantifier-free formula with CASs and obtain the answer. The time complexity of RCF-QE is quite high; it is doubly exponential in the number of variables [5]. Hence, the simplification of the formula *before* RCF-QE is a crucial step.

3 Algebraically Indexed Types

This section summarizes the framework of AIT. We refrain from presenting it in full generality and describe its application to geometry ([4, §2]) with the restriction we made on it in incorporating it into the type system of ToroboMath.

In AIT, some of the primitive types have associated *indices*. An index represents a transformation on the object of that type. For instance, in $\texttt{Vec}\langle B, t\rangle$, the index B stands for an invertible linear transformation and t stands for a translation. The index variables bound by universal quantifiers signify that a function of that type is invariant under any transformations indicated by the indices, e.g.,

$$\texttt{midpoint} : \forall B{:}\mathsf{GL}_2.\forall t{:}\mathsf{T}_2.\ \texttt{Vec}\langle B, t\rangle \rightarrow \texttt{Vec}\langle B, t\rangle \rightarrow \texttt{Vec}\langle B, t\rangle.$$

The type of `midpoint` certifies that, when two points P and Q undergo an arbitrary affine transformation, the midpoint of P and Q moves accordingly.

3.1 Sort and Index Expression

The *sort* of an index signifies the kind of transformations represented by the index. We assume the set Sort of index sorts includes GL_k ($k = 1, 2, 3$) (general linear transformations), O_k ($k = 2, 3$) (orthogonal transformations), and T_k ($k = 2, 3$) (translations). In the type of `midpoint`, B is of sort GL_2 and t is of sort T_2.

An *index expression* is composed of index variables and index operators. In the current paper, we use the following operators: $\langle +, -, 0 \rangle$ are addition, negation, and unit of T_k $(k = 2, 3)$; $\langle \cdot\,, {}^{-1}, 1 \rangle$ are multiplication, inverse, and unit of GL_k and O_k; det is the determinant; $|\cdot|$ is the absolute value. An *index context* Δ is a list of index variables paired with their sorts: $\Delta = i_1{:}S_1, i_2{:}S_2, \ldots, i_n{:}S_n$. The well-sortedness of an index expression e of sort S, written $\Delta \vdash e : S$, is defined analogously to the well-typedness in simple type theory.

3.2 Type, Term, and Typing Judgement

The set of primitive types, $\textsc{PrimType} = \{\mathtt{Bool}, \mathtt{R}, \mathtt{2d.Vec}, \mathtt{3d.Vec}, \mathtt{2d.Shape}, \ldots\}$, is the same as that in the language of ToroboMath. A function tyArity: $\textsc{PrimType} \to \textsc{Sort}^*$ specifies the number and sorts of indices appropriate for the primitive types: e.g., $\mathrm{tyArity}(\mathtt{2d.Vec}) = (\mathsf{GL}_2, \mathsf{T}_2)$.

A judgement $\Delta \vdash A$ type means that type A is well-formed and well-indexed with respect to an index context Δ. Here are the derivation rules:

$$\frac{\mathtt{X} \in \textsc{PrimType} \quad \mathrm{tyArity}(\mathtt{X}) = (S_1, \ldots, S_m) \quad \{\Delta \vdash e_j : S_j\}_{1 \le j \le m}}{\Delta \vdash \mathtt{X}\langle e_1, \ldots, e_m \rangle \ \mathsf{type}}\ \textsc{TyPrim}$$

$$\frac{\Delta \vdash A \ \mathsf{type} \quad \Delta \vdash B \ \mathsf{type}}{\Delta \vdash A \to B \ \mathsf{type}}\ \textsc{TyArr} \qquad \frac{\Delta, i{:}S \vdash A \ \mathsf{type}}{\Delta \vdash \forall i{:}S.A \ \mathsf{type}}\ \textsc{TyForall}$$

While Atkey et al.'s system is formulated in the style of System F, we allow the quantifiers only at the outermost (prenex) position. The restriction permits an efficient type reconstruction algorithm analogous to Hindley-Milner's, while being expressive enough to capture the invariance of the pre-defined functions in ToroboMath and the invariance in the majority of math problems.

The well-typedness of a term M, written $\Delta; \Gamma \vdash M : A$, is judged with respect to an index context Δ and a typing context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$. A typing context is a list of variables with their types. A special context Γ_{ops} consists of the pre-defined symbols and their types, e.g., $+ : \forall s{:}\mathsf{GL}_1.\ \mathtt{R}\langle s \rangle \to \mathtt{R}\langle s \rangle \to \mathtt{R}\langle s \rangle \in \Gamma_{\mathrm{ops}}$. We assume Γ_{ops} is always available in the typing derivation and suppress it in a judgement. The typing rules are analogous to those for lambda calculus with rank-1 polymorphism except for TyEQ:

$$\frac{x : A \in \Gamma}{\Delta; \Gamma \vdash x : A}\,\textsc{Var} \quad \frac{\Delta; \Gamma \vdash M : \forall i{:}S.A \quad \Delta \vdash e{:}S}{\Delta; \Gamma \vdash M : A\{i \mapsto e\}}\ \textsc{UnivInst} \quad \frac{\Delta; \Gamma, x : A \vdash M : B}{\Delta; \Gamma \vdash \lambda x.M : A \to B}\,\textsc{Abs}$$

$$\frac{\Delta; \Gamma \vdash M : A \to B \quad \Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash MN : B}\,\textsc{App} \quad \frac{\Delta; \Gamma \vdash M : A \quad \Delta \vdash A \equiv B}{\Delta; \Gamma \vdash M : B}\ \textsc{TyEQ}$$

In the Abs and App rules, the meta-variables A and B only designate a type without quantifiers. In the UnivInst rule, $A\{i \mapsto e\}$ is the result of substituting e for i in A. The 'polymorphism' of the types with quantifiers hence takes place only when a pre-defined symbol (e.g., `midpoint`) enters a derivation via the Var rule and then the bound index variable is instantiated via the UnivInst rule.

The type equivalence judgement $\varDelta \vdash A \equiv B$ in the TyEQ rule equates two types involving *semantically* equivalent index expressions; thus, e.g., $s{:}\mathsf{GL}_1 \vdash \mathtt{R}\langle s \cdot s^{-1}\rangle \equiv \mathtt{R}\langle 1\rangle$ and $O{:}\mathsf{O}_2 \vdash \mathtt{R}\langle |\det O|\rangle \equiv \mathtt{R}\langle 1\rangle$.

3.3 Index Erasure Semantics and Transformational Interpretation

The abstraction theorem for AIT [4] enables us to know the invariance of a term by its type. The theorem relates two kinds of interpretations of types and terms: index erasure semantics and relational interpretations. We will restate the theorem with what we here call *transformational* interpretations (t-interpretations hereafter), instead of the relational interpretations. It suffices for the purpose of justifying our algorithm and makes it easier to grasp the idea of the theorem.

The index-erasure semantics of a primitive type $\mathtt{X}\langle e_1, \ldots, e_n\rangle$ is determined only by $\mathtt{X}$. We thus write $\lfloor \mathtt{X}\langle e_1, \ldots, e_n\rangle \rfloor = \lfloor \mathtt{X} \rfloor$. The interpretation $\lfloor \mathtt{X} \rfloor$ is the set of mathematical objects intended for the type: e.g., $\lfloor \mathtt{2d.Vec}\langle B, t\rangle \rfloor = \lfloor \mathtt{2d.Vec} \rfloor = \mathbb{R}^2$ and $\lfloor \mathtt{R}\langle s\rangle \rfloor = \lfloor \mathtt{R} \rfloor = \mathbb{R}$. The index-erasure semantics of a non-primitive type is determined by the type structure: $\lfloor A \to B \rfloor = \lfloor A \rfloor \to \lfloor B \rfloor$ and $\lfloor \forall i{:}S.\ T \rfloor = \lfloor T \rfloor$.

The index-erasure semantics of a typing context $\varGamma = x_1{:}\mathtt{T}_1, \ldots, x_n{:}\mathtt{T}_n$ is the direct product of the domains of the variables: $\lfloor \varGamma \rfloor = \lfloor \mathtt{T}_1 \rfloor \times \cdots \times \lfloor \mathtt{T}_n \rfloor$. The erasure semantics of a term $\varDelta; \varGamma \vdash M : A$ is a function of the values assigned to its free variables: $\lfloor M \rfloor : \lfloor \varGamma \rfloor \to \lfloor A \rfloor$ and defined as usual (see, e.g., [18,21]).

The t-interpretation of a type $\mathtt{T}$, denoted by $[\![\mathtt{T}]\!]$, is a function from the assignments to the index variables to a transformation on $\lfloor \mathtt{T} \rfloor$. To be precise, we first define the semantics of index context $\varDelta = i_1{:}S_1, \ldots, i_n{:}S_n$ as the direct product of the interpretation of the sorts: $[\![\varDelta]\!] = [\![S_1]\!] \times \cdots \times [\![S_n]\!]$, where $[\![S_1]\!], \ldots, [\![S_n]\!]$ are the intended sets of transformations: e.g., $[\![\mathsf{GL}_2]\!] = \mathrm{GL}_2$ and $[\![\mathsf{T}_2]\!] = \mathrm{T}_2$. The interpretation of an index expression e of sort S is a function $[\![e]\!] : [\![\varDelta]\!] \to [\![S]\!]$ that is determined by the structure of the expression; for $\rho \in [\![\varDelta]\!]$,

$$[\![\mathtt{f}(e_1, \ldots, e_n)]\!](\rho) = [\![\mathtt{f}]\!]([\![e_1]\!](\rho), \ldots, [\![e_n]\!](\rho)), \quad [\![i_k]\!](\rho) = \rho(i_k),$$

where, in the last equation, we regard $\rho \in [\![\varDelta]\!]$ as a function from index variables to their values. The index operations det and $|\cdot|$ are interpreted as intended.

The t-interpretation of a primitive type $\mathtt{X}\langle e_1, \ldots, e_n\rangle$ is then determined by $\mathtt{X}$ and the structures of the index expressions $e_1, \ldots, e_n$. The t-interpretation of $\mathtt{Vec}$ and $\mathtt{Shape}$ is the affine transformation of vectors and geometric objects parametrized by $\rho \in [\![\varDelta]\!]$; for index expressions $\beta{:}\mathsf{GL}_2$ and $\tau{:}\mathsf{T}_2$,

$$\begin{aligned} [\![\mathtt{Vec}\langle \beta, \tau\rangle]\!](\rho) &: \mathbb{R}^2 \ni x \mapsto M_{[\![\beta]\!](\rho)} x + v_{[\![\tau]\!](\rho)} \in \mathbb{R}^2 \\ [\![\mathtt{Shape}\langle \beta, \tau\rangle]\!](\rho) &: \mathcal{P}(\mathbb{R}^2) \ni S \mapsto \{M_{[\![\beta]\!](\rho)} x + v_{[\![\tau]\!](\rho)} \mid x \in S\} \in \mathcal{P}(\mathbb{R}^2), \end{aligned}$$

where $M_{[\![\beta]\!](\rho)}$ and $v_{[\![\tau]\!](\rho)}$ are the representation matrix and vector of $[\![\beta]\!](\rho)$ and $[\![t]\!](\rho)$, and $\mathcal{P}(\mathbb{R}^2)$ denotes the power set of $\mathbb{R}^2$. Similarly, for the real numbers,

$$[\![\mathtt{R}\langle \sigma\rangle]\!](\rho) : \mathbb{R} \ni x \mapsto [\![\sigma]\!](\rho) x \in \mathbb{R}.$$

That is, $[\![\mathtt{R}\langle\sigma\rangle]\!](\rho)$ is a change of scale with the scaling factor determined by the expression $\sigma{:}\mathsf{GL}_1$ and the assignment ρ. For a primitive type $\mathtt{X}$ with no indices, its t-interpretation is the identity map on $\lfloor\mathtt{X}\rfloor$: i.e., $[\![\mathtt{X}]\!](\rho) = \mathrm{id}_{\lfloor X\rfloor}$.

The t-interpretation of a function type $A \to B$ is a higher-order function that maps a (mathematical) function $f : \lfloor A\rfloor \to \lfloor B\rfloor$ to another function on the same domain and codomain such that: $[\![A \to B]\!](\rho)(f) = [\![B]\!](\rho) \circ f \circ ([\![A]\!](\rho))^{-1}$. It is easy to check that this interpretation is compatible with currying. Equivalently, we may say that if $g = [\![A \to B]\!](\rho)(f)$, then f and g are in the commutative relation $g \circ [\![A]\!](\rho) = [\![B]\!](\rho) \circ f$. The typing derivation in AIT is a way to 'pull out' the effect of transformation $[\![A]\!](\rho)$ on a free variable deep inside a term by combining such commutative relations.

The t-interpretation of a fully-quantified type is the identity map on its erasure semantics: $[\![\forall i_1{:}S_1. \ldots \forall i_n{:}S_n.\ T]\!] = \mathrm{id}_{\lfloor T\rfloor}$. We don't define that of partially-quantified types because we don't need it to state the abstraction theorem.

3.4 Abstraction Theorem

The abstraction theorem for AIT enables us to detect the invariance of (the erasure-semantics of) a term under a certain set of transformations on its free variables. We first define the t-interpretation of the typing context $\Gamma = x_1 : T_1, \ldots, x_n : T_n$ as a simultaneous transformation of $\eta = (v_1, \ldots, v_n) \in \lfloor\Gamma\rfloor$:

$$[\![\Gamma]\!](\rho) : \lfloor\Gamma\rfloor \ni \eta \mapsto [\![\Gamma]\!](\rho) \circ \eta = ([\![T_1]\!](\rho) \circ v_1, \ldots, [\![T_n]\!](\rho) \circ v_n) \in \lfloor\Gamma\rfloor\,.$$

We now present a version of the abstraction theorem, restricted to the case of a term of quantifier-free type and restated with the t-interpretation:

Theorem 1 *(Abstraction [4], restated using transformational interpretation).* *If A is a quantifier-free type and $\Delta;\Gamma \vdash M : A$, then for all $\rho \in [\![\Delta]\!]$ and all $\eta \in \lfloor\Gamma\rfloor$, we have $[\![A]\!](\rho) \circ \lfloor M\rfloor(\eta) = \lfloor M\rfloor([\![\Gamma]\!](\rho) \circ \eta)$.*

Here we provide two easy corollaries of the theorem. The first one is utilized to eliminate variables from a formula while preserving the equivalence.

Corollary 1. *If $\Delta; x_1 : \mathtt{T}_1, \ldots, x_n : \mathtt{T}_n \vdash \phi(x_1, \ldots, x_n) : \mathtt{Bool}$, then for all $\rho \in [\![\Delta]\!]$, we have $\phi(x_1, \ldots, x_n) \Leftrightarrow \phi([\![\mathtt{T}_1]\!](\rho) \circ x_1, \ldots, [\![\mathtt{T}_n]\!](\rho) \circ x_n)$.*

This is by the abstraction theorem and the fact $[\![\mathtt{Bool}]\!](\rho) = \mathrm{id}_{\lfloor\mathtt{Bool}\rfloor}$ for any ρ. It indicates that, without loss of generality, we may 'fix' some of the variables to, e.g., zeros by appropriately choosing ρ.

The second corollary is for providing more intuition about the theorem.

Corollary 2. *If $\epsilon;\epsilon \vdash \lambda x_1.\ \ldots \lambda x_n.\ f(x_1, \ldots, x_n) : \forall\Delta.\ \mathtt{T}_1 \to \cdots \to \mathtt{T}_n \to \mathtt{T}_0$ then, for all $\rho \in [\![\Delta]\!]$ and all $v_i \in \lfloor\mathtt{T}_i\rfloor$ $(i = 1, \ldots, n)$,*

$$[\![\mathtt{T}_0]\!](\rho) \circ \lfloor f\rfloor(v_1, \ldots, v_n) = \lfloor f\rfloor([\![\mathtt{T}_1]\!](\rho) \circ v_1, \ldots, [\![\mathtt{T}_n]\!](\rho) \circ v_n).$$

In the statement, $\forall\Delta$ signifies the universal quantification over all index variables in Δ. By this corollary, for instance, we can tell from the type of `midpoint` that, for all $x_1, x_2 \in \mathbb{R}^2$ and for all $g \in \mathrm{GL}_2$ and $t \in \mathrm{T}_2$,

$$\lfloor\mathtt{midpoint}\rfloor(M_g x_1 + v_t, M_g x_2 + v_t) = M_g \lfloor\mathtt{midpoint}\rfloor(x_1, x_2) + v_t.$$

3.5 Restriction on the Index Expressions of Sort $\mathsf{GL}_k/\mathsf{O}_k$ ($k \geq 2$)

We found that the type reconstruction in AIT is far more straightforward when we assume an index expression of sort GL_k or O_k ($k \geq 2$) includes at most one index variable of sort GL_k or O_k that is not in the determinant operator. Assuming this, any expression e of sort GL_k or O_k can be written in the form of

$$e = \prod_{i \in I} s_i^{w_i} \cdot \prod_{i \in I} |s_i|^{x_i} \cdot \prod_{j \in J} \det(B_j)^{y_j} \cdot \prod_{j \in J} |\det(B_j)|^{z_j} \cdot B_0^{\delta},$$

where $\{s_i\}_{i \in I}$ are of sort GL_1, $\{B_0\} \cup \{B_j\}_{j \in J}$ are of sort GL_k or O_k, $w_i, x_i, y_j, z_j \in \mathbb{Z}$, and $\delta \in \{0, 1\}$. We henceforth say an expression e in the above form satisfies *the head variable property* and call B_0 the *head variable* of e.

Empirically, this restriction is not too restrictive; as far as we are aware of, the invariance of all the pre-defined functions and predicates in ToroboMath is expressible with an indexed-type satisfying this.

4 Invariance Detection Through Type Reconstruction

We need type reconstruction in AIT for two purposes: to infer the invariance of the pre-defined symbols in ToroboMath and to infer the invariance in a math problem. To this end, we only have to derive the judgement $\varDelta; \varGamma \vdash \phi : \mathtt{Bool}$ where ϕ is either a defining axiom of a symbol or a formula of a problem. For a pre-defined symbol s, by a judgement $\varDelta; s : T, \cdots \vdash \phi : \mathtt{Bool}$, we know s is of type T and it has the invariance signified by T. For a problem ϕ, by the judgement $\varDelta; x_1 : T_1, \ldots, x_n : T_n \vdash \phi : \mathtt{Bool}$, we know the invariance of ϕ under the transformation on the free variables $x_1, \ldots, x_n$ according to $[\![T_1]\!], \ldots, [\![T_n]\!]$.

Since all types are in prenex form, we can find the typing derivation by a procedure analogous to the Hindley-Milner (H-M) algorithm. It consists of two steps: deriving equations among index expressions, and solving them. The procedure for solving the equations in $\mathsf{T}_2/\mathsf{T}_3$ is essentially the same as in the type inference for Kennedy's unit-of-measure types [11], which is a precursor of AIT. Further development is required to solve the equations in $\mathsf{GL}_2/\mathsf{GL}_3$, even under the restriction on the form of index expressions mentioned in Sect. 3.5, due to the existence of the index operations $|\cdot|$ and det.

4.1 Equation Derivation

We first assign a type variable α_i for each subterm t_i in ϕ. Then, for a subterm t_i in the form $t_j t_k$ (i.e., application of t_j to t_k), we have the equation $\alpha_j = \alpha_k \to \alpha_i$. The case for a subterm t_i in the form of $\lambda x.t_j$ is also analogous to H-M and we omit it here. For a leaf term (i.e., a variable) t_i, if it is one of the pre-defined symbols and $t_i : \forall i_1{:}S_1. \ldots \forall i_n{:}S_n.T \in \varGamma_{\mathrm{ops}}$, we set $\alpha_i = T\{i_1 \mapsto \beta_1, \ldots, i_n \mapsto \beta_n\}$, where $\{i_1 \mapsto \beta_1, \ldots, i_n \mapsto \beta_n\}$ stands for the substitution of fresh variables $\beta_1, \ldots, \beta_n$ for $i_1, \ldots, i_n$. By solving the equations for the type and index variables $\{\alpha_i\}$ and $\{\beta_j\}$, we reconstruct the most general indexed-types of all the subterms.

For example, consider the following axiom defining $\texttt{perpendicular}$:

$$\forall v_1.\forall v_2.(\texttt{perpendicular}(v_1, v_2) \longleftrightarrow \texttt{inner-prod}(v_1, v_2) = 0),$$

and suppose that $\texttt{inner-prod}$ is in Γ_{ops}. We are going to reconstruct the type of $\texttt{perpendicular}$. The type of $\texttt{inner-prod}$ is

$$\texttt{inner-prod} : \forall s_1, s_2{:}\mathsf{GL}_1.\ \forall O{:}\mathsf{O}_2.\ \texttt{Vec}\langle s_1 O, 0\rangle \to \texttt{Vec}\langle s_2 O, 0\rangle \to \texttt{R}\langle s_1 \cdot s_2\rangle$$

and it is instantiated as $\texttt{inner-prod} : \texttt{Vec}\langle s_1 O, 0\rangle \to \texttt{Vec}\langle s_2 O, 0\rangle \to \texttt{R}\langle s_1 \cdot s_2\rangle$ where s_1, s_2, and O are fresh variables. Since the type of $\texttt{perpendicular}$ in the non-AIT version of our language is $\texttt{Vec} \to \texttt{Vec} \to \texttt{Bool}$, we set fresh variables to all indices in the primitive types and have:

$$\texttt{perpendicular} : \texttt{Vec}\langle \beta_1, \tau_1\rangle \to \texttt{Vec}\langle \beta_2, \tau_2\rangle \to \texttt{Bool}.$$

Since $\texttt{perpendicular}$ is applied to v_1 and v_2, the types of v_1 and v_2 are equated to $\texttt{Vec}\langle \beta_1, \tau_1\rangle$ and $\texttt{Vec}\langle \beta_2, \tau_2\rangle$. Additionally, since $\texttt{inner-prod}$ is also applied to v_1 and v_2, we have the following equations:

$$\texttt{Vec}\langle s_1 O, 0\rangle = \texttt{Vec}\langle \beta_1, \tau_1\rangle, \quad \texttt{Vec}\langle s_2 O, 0\rangle = \texttt{Vec}\langle \beta_2, \tau_2\rangle \tag{4.1}$$

If we have an equation between the same primitive type, by unifying both sides of the equation, in turn we have one or more equations between index expressions, i.e., if we have $\texttt{X}\langle e_1, \ldots, e_m\rangle = \texttt{X}\langle e'_1, \ldots, e'_m\rangle$, then we have: $e_1 = e'_1, \ldots, e_m = e'_m$. For Eq. (4.1), we hence have $s_1 O = \beta_1, s_2 O = \beta_2, 0 = \tau_1$, and $0 = \tau_2$. Thus, by recursively unifying all the equated types, we are left with a system of equations between index expressions.

4.2 Equation Solving

To solve the derived equations between index expressions, we need to depart from the analogy with the H-M algorithm. Namely, instead of applying syntactic unification, we need semantic unification, i.e., we solve the equations as simultaneous equations in the transformation groups.

We first order the equations with respect to the sort of the equated expressions. We then process them in the order $\mathsf{T}_2/\mathsf{T}_3 \to \mathsf{GL}_2/\mathsf{GL}_3 \to \mathsf{GL}_1$ as follows.[1]

First, since equations of sort $\mathsf{T}_2/\mathsf{T}_3$ are always in the form of $\sum_i a_i t_i = 0$ ($a_i \in \mathbb{Z}$), where $\{t_i\}$ are variables of sort T_k ($k \in \{2, 3\}$), we can solve the equations as is the case with a linear homogeneous system. Although the solution may involve rational coefficients as in $t_i = \sum_j \frac{n_{ij}}{m_{ij}} t_j$ ($n_{ij}, m_{ij} \in \mathbb{Z}$), we can clear the denominators by introducing new variables t'_j such that $t_j = \text{lcm}\{m_{ij}\}_i \cdot t'_j$.

Next, by the head variable property, equations of sort $\mathsf{GL}_2/\mathsf{GL}_3$ (henceforth $\mathsf{GL}_{\geq 2}$) are always in the form of $\sigma_1 B_1 = \sigma_2 B_2$, where σ_1 and σ_2 are index

[1] In this subsection, $\mathsf{GL}_2, \mathsf{GL}_3, \mathsf{O}_2$, and O_3 are collectively denoted as $\mathsf{GL}_2/\mathsf{GL}_3$ or $\mathsf{GL}_{\geq 2}$.

expressions of sort GL_1, and B_1 and B_2 are the head variables of sort $\mathsf{GL}_{\geq 2}$. We decompose these equations according to Table 2, which summarizes the following argument: Let E denote the identity transformation. Since $\sigma_1 B_1 = \sigma_2 B_2 \iff \sigma_1^{-1}\sigma_2 E = B_1 B_2^{-1}$, there must be some $s \in \mathsf{GL}_1$ such that $B_1 B_2^{-1} = sE$ and $\sigma_1^{-1}\sigma_2 = s$. Furthermore, by the superset-subset relation between the sorts of B_1 and B_2, e.g., $\mathrm{O}_2 \subset \mathrm{GL}_2$ for $B_1 : \mathsf{O}_2$ and $B_2 : \mathsf{GL}_2$, we can express one of the broader sort with the other as a parameter.

The algorithm for $\mathsf{GL}_{\geq 2}$ equations works as follows. First, we initialize the set of solution with the empty substitution: $S \leftarrow \{\}$. For each $\mathsf{GL}_{\geq 2}$ equation $\sigma_1 B_1 = \sigma_2 B_2$, we look up Table 2 and find the $\mathsf{GL}_{\geq 2}$ solution $B_i \mapsto sB_j$ and one or more new GL_1 equations. We populate the current set of GL_1 equations with the new ones, and apply the solution $B_i \mapsto sB_j$ to all the remaining GL_1 and $\mathsf{GL}_{\geq 2}$ equations. We also compose the GL_2 solution $B_i \mapsto sB_j$ with the current solution set: $S \leftarrow S \circ \{B_i \mapsto sB_j\}$.

By processing all $\mathsf{GL}_{\geq 2}$ equations as above, we are left with a partial solution S and a system of GL_1 equations, each of which is in the following form:

$$\prod_{i\in I} s_i^{w_i} \cdot \prod_{i\in I} |s_i|^{x_i} \cdot \prod_{j\in J} \det(B_j)^{y_j} \cdot \prod_{j\in J} |\det(B_j)|^{z_j} = 1 \quad (w_i, x_i, y_j, z_j \in \mathbb{Z}),$$

where we assume about I and J that $\{s_i\}_{i\in I}$ are all the GL_1 variables, $\{B_j\}_{j\in J}$ are all the remaining $\mathsf{GL}_{\geq 2}$ variables, and $I \cap J = \emptyset$. Letting $u_i = s_i \cdot |s_i|^{-1}$, $v_i = |s_i|$, $u_j = \det(B_j) \cdot |\det(B_j)|^{-1}$, and $v_i = |\det(B_j)|$, we have $s_i = u_i v_i$ and $\det(B_j) = u_j v_j$ for all $i \in I$ and $j \in J$. By using them, we have

$$\prod_i u_i^{w_i} \cdot \prod_i v_i^{w_i + x_i} \cdot \prod_j u_j^{y_j} \cdot \prod_j v_j^{y_j + z_j} = 1.$$

Since $u_i, u_j \in \{+1, -1\}$ and $v_i, v_j > 0$ for all i and j, we know the above equation is equivalent to the following two equations:

$$\prod_i u_i^{w_i} \cdot \prod_j u_j^{y_j} = 1, \qquad \prod_i v_i^{w_i + x_i} \cdot \prod_j v_j^{y_j + z_j} = 1.$$

We thus have two systems of equations, one in $\{+1, -1\}$ and the other in $\mathbb{R}_{>0}$. Now we temporarily rewrite the solution with u_i and v_i: $S \leftarrow S \circ \{s_i \mapsto u_i v_i\}_{i\in I}$.

First consider the system in $\mathbb{R}_{>0}$. As long as there remains an equation involving a variable v_i, which originates from a GL_1 variable, we solve it for v_i and compose the solution $v_i \mapsto \prod_{i' \neq i} v_{i'}^{p_{i'}} \cdot \prod_j v_j^{q_j}$ with S while applying it to the remaining equations. The denominators of fractional exponents (i.e., $p_{i'}, q_j \in \mathbb{Q} \backslash \mathbb{Z}$) can be cleared similarly to the case of T_k equations. If all the equations in $\mathbb{R}_{>0}$ are solved this way, then S is the most general solution. Otherwise, there remain one or more equations of the form $\prod_{j\in J'} |\det B_j|^{d_j} = 1$ for some $J' \subset J$ and $\{d_j\}_{j\in J'}$. This is the only case where we may miss some invariance of a formula; in general, we cannot express the most general solution to this equation only with the index variables of sort GL_k and O_k. We make a compromise here and are satisfied with a less general solution $S \circ \{B_j \mapsto E\}_{j\in J'}$. Fortunately, this

Table 2. Decomposition of $\mathsf{GL}_2/\mathsf{GL}_3$ equation $\sigma_i B_i = \sigma_j B_j$ (s: a fresh variable)

Combination of head variables	Solution in GL_k	Equations in GL_1
$B_i = B_j$	none	$\sigma_i = \sigma_j$
$B_i : \mathsf{O}_k \ \wedge \ B_j = E$	$B_i \mapsto sE$	$s\sigma_i = \sigma_j, \ \lvert s\rvert = 1$
$B_i : \mathsf{GL}_k \ \wedge \ B_j = E$	$B_i \mapsto sE$	$s\sigma_i = \sigma_j$
$B_i : \mathsf{O}_k \ \wedge \ B_j : \mathsf{O}_k$	$B_i \mapsto sB_j$	$s\sigma_i = \sigma_j, \ \lvert s\rvert = 1$
$B_i : \mathsf{GL}_k \ \wedge \ B_j : \mathsf{O}_k$	$B_i \mapsto sB_j$	$s\sigma_i = \sigma_j$
$B_i : \mathsf{GL}_k \ \wedge \ B_j : \mathsf{GL}_k$	$B_i \mapsto sB_j$	$s\sigma_i = \sigma_j$

does not frequently happen in practice. We made this compromise only on three out of 533 problems used in the experiment. We expect that having more sorts, e.g., $\mathrm{SL}_k^{\pm} = \{M \in \mathrm{GL}_k \mid |\det M| = 1\}$, in the language of index expressions might be of help here, but leave it as a future work.

The system in $\{+1, -1\}$ is processed analogously to that in $\mathbb{R}_{>0}$. Finally, by restoring $\{u_i, v_i\}_{i\in I}$ and $\{u_j, v_j\}_{j\in J}$ in the solution S to their original forms, e.g., $u_i \mapsto s_i \cdot |s_i|^{-1}$, we have a solution to the initial set of equations in terms of the variables of sort GL_k and O_k.

4.3 Type Reconstruction for Pre-defined Symbols with Axioms

We incrementally determined the indexed-types of the pre-defined symbols according to the hierarchy of their definitions. We first constructed a directed acyclic graph wherein the nodes are the pre-defined symbols and the edges represent the dependency between their definitions. We manually assigned an indexed-type to the symbols without defining axioms (e.g., $+ : \mathtt{R} \to \mathtt{R} \to \mathtt{R}$) and initialized Γ_{ops} with them. We then reconstructed the indexed-types of other symbols in a topological order of the graph. After the reconstruction of the type of each symbol, we added the symbol with its inferred type to Γ_{ops}.

For some of the symbols, type reconstruction does not go as well as we hope. For example, the following axiom defines the symbol `midpoint`:

$$\forall p_1, p_2.(\mathtt{midpoint}(p_1, p_2) = \frac{1}{2} \cdot (p_1 \mathbin{\texttt{+}} p_2)).$$

At the beginning of the type reconstruction of `midpoint`, the types of the symbols in the axiom are instantiated as follows:

$$\begin{aligned} \mathtt{midpoint} &: \mathtt{Vec}\langle \beta_1, \tau_1\rangle \to \mathtt{Vec}\langle \beta_2, \tau_2\rangle \to \mathtt{Vec}\langle \beta_3, \tau_3\rangle \\ \cdot &: \mathtt{R}\langle s_1\rangle \to \mathtt{Vec}\langle B_1, 0\rangle \to \mathtt{Vec}\langle s_1 B_1, 0\rangle \\ \texttt{+} &: \mathtt{Vec}\langle B_2, t_1\rangle \to \mathtt{Vec}\langle B_2, t_2\rangle \to \mathtt{Vec}\langle B_2, t_1 + t_2\rangle. \end{aligned}$$

The derived equations between the index expressions are as follows:

$$\{B_2 = \beta_1, B_2 = \beta_2, B_1 = B_2, \beta_3 = s_1 B_1, s_1 = 1, t_1 = \tau_1, t_2 = \tau_2, 0 = t_1 + t_2, \tau_3 = 0\}.$$

By solving these equations, we obtain the indexed-type of `midpoint` as follows:

$$\texttt{midpoint} : \forall B_1{:}\mathsf{GL}_2.\ \forall t_1{:}\mathsf{T}_2.\ \texttt{Vec}\langle B_1, t_1\rangle \to \texttt{Vec}\langle B_1, -t_1\rangle \to \texttt{Vec}\langle B_1, 0\rangle.$$

This type indicates that the midpoint of any two points P and Q remains the same when we move P and Q respectively to $P + t_1$ and $Q - t_1$ for any $t_1 \in \mathbb{R}^2$. While it is *not wrong*, the following type is more useful for our purpose:

$$\texttt{midpoint} : \forall B{:}\mathsf{GL}_2.\ \forall t{:}\mathsf{T}_2.\ \texttt{Vec}\langle B, t\rangle \to \texttt{Vec}\langle B, t\rangle \to \texttt{Vec}\langle B, t\rangle. \qquad (1)$$

To such symbols, we manually assigned a more appropriate type.[2]

In the current system, 945 symbols have a type that includes indices. We manually assigned the types to 255 symbols that have no defining axioms. For 203 symbols we manually overwrote the inferred type as in the case of `midpoint`. The types of the remaining 487 symbols were derived through the type reconstruction.

5 Variable Elimination Based on Invariance

In this section, we first provide an example of the variable elimination procedure based on invariance. We then describe the top-level algorithm of the variable elimination, which takes a formula as input and eliminates some of the quantified variables in it by utilizing the invariance indicated by an index variable. We finally list the elimination rule for each sort of index variable.

5.1 Example of Variable Elimination Based on Invariance

Let us consider again the proof of the existence of the centroid of a triangle. For triangle ABC, the configuration of the midpoints P, Q, R of the three sides and the centroid G is described by the following formula:

$$\psi(A, B, C, P, Q, R, G) := \begin{pmatrix} P = \texttt{midpoint}(B, C) \land \texttt{on}(G, \texttt{segment}(A, P)) \land \\ Q = \texttt{midpoint}(C, A) \land \texttt{on}(G, \texttt{segment}(B, Q)) \land \\ R = \texttt{midpoint}(A, B) \land \texttt{on}(G, \texttt{segment}(C, R)) \end{pmatrix}$$

where $\texttt{on}(X, Y)$ stands for the inclusion of point X in a geometric object Y, and $\texttt{segment}(X, Y)$ stands for the line segment between points X and Y. Let ϕ denote the existence of the centroid (and the three midpoints):

$$\phi(A, B, C) := \exists G.\ \exists P.\ \exists Q.\ \exists R.\ \psi(A, B, C, P, Q, R, G).$$

Our goal is to prove $\forall A.\ \forall B.\ \forall C.\ \phi(A, B, C)$.

[2] The awkwardness of the type inferred for `midpoint` is a price for the efficiency of type reconstruction; it is due to the fact that we ignore the linear space structure of T_2 (and also, we do not posit $\mathrm{T}_1 (\simeq \mathbb{R})$ as the second index of type `R`). Otherwise, the type reconstruction comes closer to a search for an invariance on the algebraic representation of the problems and the defining axioms. Hence $1/2 * (t + t) = t$ is not deduced for $t : \mathsf{T}_2$, which is necessary to infer the type in Eq. (1).

The functions `midpoint`, `on`, and `segment` are invariant under translations and general linear transformations. The reconstruction algorithm hence derives

$$\beta : \mathsf{GL}_2, \tau : \mathsf{T}_2 \ ; \ A : \mathtt{Vec}\langle \beta, \tau\rangle, B : \mathtt{Vec}\langle \beta, \tau\rangle, C : \mathtt{Vec}\langle \beta, \tau\rangle \vdash \phi(A, B, C) : \mathtt{Bool}.$$

By the abstraction theorem, this judgement implies the invariance of the proposition $\phi(A, B, C)$ under arbitrary affine transformations:

$$\forall g \in \mathrm{GL}_2.\ \forall t \in \mathrm{T}_2.\ \forall A, B, C.\ \phi(A, B, C) \Leftrightarrow \phi(t \circ g \circ A, t \circ g \circ B, t \circ g \circ C).$$

First, by considering the case of g being identity, we have

$$\forall t \in \mathrm{T}_2.\ \forall A, B, C.\ \phi(A, B, C) \Leftrightarrow \phi(t \circ A, t \circ B, t \circ C). \tag{2}$$

By using this, we are going to verify $\forall B, C.\ \phi(\mathbf{0}, B, C) \Leftrightarrow \forall A, B, C.\ \phi(A, B, C)$, by which we know that we only have to prove $\forall B, C.\ \phi(\mathbf{0}, B, C)$.

Suppose that $\forall B, C.\ \phi(\mathbf{0}, B, C)$ holds. Since T_2 acts transitively on $\mathbb{R}^2$, for any $A \in \mathbb{R}^2$, there exists $t \in \mathrm{T}_2$ such that $t \circ \mathbf{0} = A$. Furthermore, for any $B, C \in \mathbb{R}^2$, by instantiating $\forall B, C.\ \phi(\mathbf{0}, B, C)$ with $B \mapsto t^{-1} \circ B$ and $C \mapsto t^{-1} \circ C$, we have $\phi(\mathbf{0}, t^{-1} \circ B, t^{-1} \circ C)$. By Eq. (2), we obtain $\phi(t \circ \mathbf{0}, t \circ t^{-1} \circ B, t \circ t^{-1} \circ C)$, which is equivalent to $\phi(A, B, C)$. Since A, B, C were arbitrary, we proved

$$\forall B, C.\ \phi(\mathbf{0}, B, C) \Rightarrow \forall A, B, C.\ \phi(A, B, C).$$

The converse is trivial. We thus proved $\forall B, C.\ \phi(\mathbf{0}, B, C) \Leftrightarrow \forall A, B, C.\ \phi(A, B, C)$.

The simplified formula, $\forall B, C.\ \phi(\mathbf{0}, B, C)$, is still invariant under the simultaneous action of GL_2 on B and C. Hence, by applying the type reconstruction again, we have $\beta : \mathsf{GL}_2 \ ; \ B : \mathtt{Vec}\langle \beta, 0\rangle, C : \mathtt{Vec}\langle \beta, 0\rangle \vdash \phi(\mathbf{0}, B, C) : \mathtt{Bool}$. It implies the following invariance: $\forall g \in \mathrm{GL}_2.\ \forall B, C.\ \phi(\mathbf{0}, B, C) \Leftrightarrow \phi(\mathbf{0}, g \circ B, g \circ C)$.

We now utilize it to eliminate the remaining variables B and C. Although it is tempting to 'fix' B and C respectively at, e.g., $\mathbf{e}_1 := (1, 0)$ and $\mathbf{e}_2 := (0, 1)$, it incurs some *loss* of generality. For instance, when B is at the origin, there is no way to move B to $\mathbf{e}_1$ by any $g \in \mathrm{GL}_2$. We consider four cases:

1. B and C are linearly independent,
2. $B \neq \mathbf{0}$, and B and C are linearly dependent,
3. $C \neq \mathbf{0}$, and B and C are linearly dependent, and
4. B and C are both at the origin.

For each of these cases, we can find a suitable transformation in GL_2 as follows:

1. There exists $g_1 \in \mathrm{GL}_2$ s.t. $g_1 \circ B = \mathbf{e}_1$ and $g_1 \circ C = \mathbf{e}_2$,
2. There exist $g_2 \in \mathrm{GL}_2$ and $r \in \mathbb{R}$ s.t. $g_2 \circ B = \mathbf{e}_1$ and $g_2 \circ C = r\mathbf{e}_1$,
3. There exist $g_3 \in \mathrm{GL}_2$ and $r' \in \mathbb{R}$ s.t. $g_3 \circ C = \mathbf{e}_1$ and $g_3 \circ B = r'\mathbf{e}_1$, and
4. We only have to know whether or not $\phi(\mathbf{0}, \mathbf{0}, \mathbf{0})$ holds.

By a similar argument to the one for the translation-invariance, we have

$$\forall B, C.\ \phi(\mathbf{0}, B, C) \Leftrightarrow \phi(\mathbf{0}, \mathbf{e}_1, \mathbf{e}_2) \wedge \forall r.\ \phi(\mathbf{0}, \mathbf{e}_1, r\mathbf{e}_1) \wedge \forall r'.\ \phi(\mathbf{0}, r'\mathbf{e}_1, \mathbf{e}_1) \wedge \phi(\mathbf{0}, \mathbf{0}, \mathbf{0}).$$

Thus, we eliminated all four coordinate values (i.e., x and y coordinates for B and C) in the first and the last case and three of them in the other two cases.

5.2 Variable Elimination Algorithm

The variable elimination algorithm works as follows. We traverse the formula of a problem in a top-down order and, for each subformula in the form of

$$Qx_1.Qx_2.\cdots Qx_n.\ \phi(x_1,x_2,\ldots,x_n,\mathbf{y})\quad (Q\in\{\forall,\exists\})$$

where $\mathbf{y}=y_1,\ldots,y_m$ are the free variables, we apply the type reconstruction procedure to $\phi(x_1,x_2,\ldots,x_n,\mathbf{y})$ and derive a judgement $\varDelta;\varGamma,x_1{:}\mathtt{T}_1,\ldots,x_n{:}\mathtt{T}_n\vdash\phi(x_1,\ldots,x_n,\mathbf{y}):\mathtt{Bool}$. We then choose an index variable i that appears at least once in $\mathtt{T}_1,\ldots,\mathtt{T}_n$ but in none of the types of $\mathbf{y}$. It means the transformation signified by i acts on some of $\{x_1,\ldots,x_n\}$ but on none of $\mathbf{y}$. We select from $\{x_1,\ldots,x_n\}$ one or more variables whose types include i and are of the form $\mathtt{R}\langle\sigma\rangle$ or $\mathtt{Vec}\langle\beta,\tau\rangle$. Suppose that we select $x_1,\ldots,x_l$. Then we know the judgement $\varDelta;\varGamma,x_1{:}\mathtt{T}_1,\ldots,x_l{:}\mathtt{T}_l\vdash Qx_{l+1}.\cdots Qx_n.\ \phi(x_1,\ldots,x_n,\mathbf{y}):\mathtt{Bool}$ also holds. We then eliminate (or add restriction on) the bound variables $x_1,\ldots,x_l$ by one of the lemmas in Sect. 5.3 according to the sort of i. After the elimination, the procedure is recursively applied to the resulting formula and its subformulas.

5.3 Variable Elimination Rules

We now present how to eliminate variables based on a judgement of the form

$$\varDelta;\varGamma,\ x_1:\mathtt{T}_1,\ldots,x_n:\mathtt{T}_n\vdash\psi(x_1,\ldots,x_n,\mathbf{y}):\mathtt{Bool}$$

where $\mathtt{T}_1,\ldots,\mathtt{T}_n$ include no other variables than i; $\varGamma=y_1{:}\mathtt{U}_1,\ldots,y_m{:}\mathtt{U}_m$ is a typing context for $\mathbf{y}=y_1,\ldots,y_m$; and $\mathtt{U}_1\ldots,\mathtt{U}_m$ do not include i. Note that we can obtain a judgement of this form by the procedure in Sect. 5.2 and by substituting the unity of appropriate sorts for all index variables other than i in $\mathtt{T}_1,\ldots,\mathtt{T}_n$.

We provide the variable elimination rules as lemmas, one for each sort of i. They state the rules for variables bound by ∀. The rules for ∃ are analogous. In stating the lemma, we suppress $\varDelta$ and $\varGamma$ in the judgement and $\mathbf{y}$ in ψ for brevity but we still assume the above-mentioned condition hold.

Some complication arises due to the fact that if $k\neq l$, then $\mathtt{T}_k$ and $\mathtt{T}_l$ may be indexed with *different* expressions of i. We thus need to consider potentially different transformations $[\![\mathtt{T}_1]\!](i),\ldots,[\![\mathtt{T}_n]\!](i)$ applied simultaneously on $x_1,\ldots,x_n$. Please refer to supplementary material on the first author's web page for a general argument behind the rules and the proofs of the lemmas (https://researchmap.jp/mtzk/?lang=en).

T_k: The following lemma states that, as we saw in Sect. 5.1, we have only to consider the truth of a formula $\psi(x)$ at $x=\mathbf{0}$ if $\psi(x)$ is translation-invariant.

Lemma 1. *If* $x:\mathtt{Vec}\langle 1,\tau(t)\rangle\vdash\psi(x):\mathtt{Bool}$ *holds for* $t:\mathsf{T}_k$ $(t\in\{2,3\})$*, then* $\forall x.\ \psi(x)\Leftrightarrow\psi(\mathbf{0})$.

O_2: The following lemma means that we may assume x is on the x-axis if $\psi(x)$ is invariant under rotation and reflection.

Lemma 2. *If* $x : \mathtt{Vec}\langle \beta(O), 0\rangle \vdash \psi(x) : \mathtt{Bool}$ *holds for* $O : \mathsf{O}_2$*, then* $\forall x.\ \psi(x) \Leftrightarrow \forall r.\ \psi(r\mathbf{e}_1)$*.*

O_3: A judgement in the following form implies different kinds of invariance according to β_1 and β_2:

$$x_1 : \mathtt{Vec}\langle \beta_1(O), 0\rangle, x_2 : \mathtt{Vec}\langle \beta_2(O), 0\rangle \vdash \psi(x_1, x_2) : \mathtt{Bool}. \tag{3}$$

In any case, we may assume x_1 is on the x-axis and x_2 is on the xy-plane for proving $\forall x_1, x_2.\ \psi(x_1, x_2)$, as stated in the following lemma.

Lemma 3. *If judgement (3) holds for* $O : \mathsf{O}_3$*, then*

$$\forall x_1.\ \forall x_2.\ \psi(x_1, x_2) \Leftrightarrow \forall p, q, r \in \mathbb{R}.\ \psi(p\mathbf{e}_1, q\mathbf{e}_1 + r\mathbf{e}_2).$$

GL_1: For $s : \mathsf{GL}_1$, a judgement $x : \mathtt{R}\langle \sigma(s)\rangle \vdash \psi(x) : \mathtt{Bool}$ implies, either

- $\psi(x)$ is invariant under change of sign, i.e., $\psi(x) \Leftrightarrow \psi(-x)$,
- $\psi(x)$ is invariant under positive scaling, i.e., $\psi(x) \Leftrightarrow \psi(fx)$ for all $f > 0$, or
- $\psi(x)$ is invariant under arbitrary scaling, i.e., $\psi(x) \Leftrightarrow \psi(fx)$ for all $f \neq 0$.

The form of σ determines the type of invariance. The following lemma summarizes how we can eliminate or restrict a variable for these cases.

Lemma 4. *Let* $\sigma(s) = s^e \cdot |s|^f \quad (e \neq 0$ *or* $f \neq 0)$ *and suppose a judgement* $x : \mathtt{R}\langle \sigma(s)\rangle \vdash \psi(s) : \mathtt{Bool}$ *holds for* $s : \mathsf{GL}_1$*. We have three cases:*

1. *if* $e + f = 0$*, then* $\forall x.\ \psi(x) \Leftrightarrow \forall x \geq 0.\ \psi(x)$*, otherwise,*
2. *if* e *is an even number, then* $\forall x.\ \psi(x) \Leftrightarrow \psi(1) \wedge \psi(0) \wedge \psi(-1)$*, and*
3. *if* e *is an odd number, then* $\forall x.\ \psi(x) \Leftrightarrow \psi(1) \wedge \psi(0)$*.*

GL_2 For $B : \mathsf{GL}_2$, a judgement in the following form implies different kinds of invariance of $\psi(x_1, x_2)$ depending on the form of β_1 and β_2:

$$x_1 : \mathtt{Vec}\langle \beta_1(B), 0\rangle, x_2 : \mathtt{Vec}\langle \beta_2(B), 0\rangle \vdash \psi(x_1, x_2). \tag{4}$$

The following lemma summarizes how we eliminate the variables in each case.

Lemma 5. *Let* $\beta_j(B) = \det(B)^{e_j} \cdot |\det(B)|^{f_j} \cdot B$ *and* $g_j = e_j + f_j$ $(j \in \{1, 2\})$*. If judgement (4) holds, then, letting* $\psi_0 := \psi(\mathbf{0}, \mathbf{0}) \wedge \forall r.\ \psi(r\mathbf{e}_1, \mathbf{e}_1) \wedge \forall r.\ \psi(\mathbf{e}_1, r\mathbf{e}_1)$ *and* $\Psi := \forall x_1.\ \forall x_2.\ \psi(x_1, x_2)$*, the following equivalences hold:*

1. *If* $g_1 + g_2 + 1 = 0$ *and*
   - *if* $e_1 + e_2$ *is an even number, then* $\Psi \Leftrightarrow \psi_0 \wedge \psi(\mathbf{e}_1, \mathbf{e}_2)$
   - *if* $e_1 + e_2$ *is an odd number, then* $\Psi \Leftrightarrow \psi_0 \wedge \psi(\mathbf{e}_1, \mathbf{e}_2) \wedge \psi(\mathbf{e}_1, -\mathbf{e}_2)$
2. *If* $g_1 + g_2 + 1 \neq 0$*, then* $\Psi \Leftrightarrow \psi_0 \wedge \forall r.\ \psi(r\mathbf{e}_1, \mathbf{e}_2)$*.*

A similar lemma holds for the invariances indicated by an index variable of sort GL_3. We refrain from presenting it for space reasons.

Table 3. Results on All RCF Problems in ToroboMath Benchmark

Division/#Prblms		AlgIdx		Baseline	
		Solved	Time	Solved	Time
IMO	116	28%	51.7s	16%	19.7s
Univ	243	69%	22.1s	62%	26.8s
Chart	174	68%	9.7s	62%	12.0s
All	533	60%	20.5s	52%	20.6s

Table 4. Results on RCF Problems with Invariance Detected and Variable Eliminated

Division/#Prblms		AlgIdx		Baseline		Speed
		Solved	Time	Solved	Time	up
IMO	77	19%	91.3s	1%	3.6s	23%
Univ	49	57%	31.0s	33%	62.7s	495%
Chart	77	49%	14.3s	36%	26.0s	529%
All	203	40%	34.3s	22%	38.5s	505%

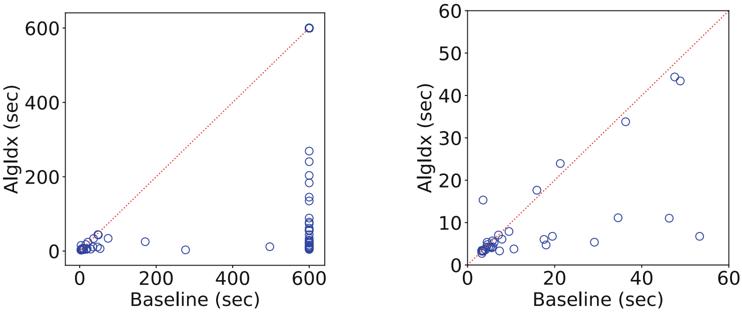


Fig. 5. Comparison of Elapsed Time with and without the Invariance Detection based on AITs (Left: All Problems; Right: Problems Solved within 60 s)

6 Experiment

We evaluated the effectiveness of the proposed method on the pre-university math problems in the ToroboMath benchmark. We used a subset of the problems that can be naturally expressible (by human) in the language of RCF. Most of them are either in geometry or algebra. Note that the formalization was done in the language introduced in Sect. 2 but not directly in the language of RCF. The problems are divided according to the source of the problems; **IMO** problems were taken from past International Mathematical Olympiads, **Univ** problems were from entrance exams of Japanese universities, and **Chart** problems were from a popular math practice book series. Please refer to another paper [16] on the ToroboMath benchmark for the details of the problems.

The type reconstruction and formula simplification procedures presented in Sect. 4 and Sect. 5 were implemented as a pre-processor of the formalized problems. The time spent for the preprocessing was almost negligible (0.76 s per problem on average) compared to that for solving the problems.

We compared the ToroboMath system with and without the pre-processor (respectively called AlgIdx and Baseline below). The Baseline system *is equipped with* Iwane and Anai's invariance detection and simplification algorithm [10] that operates on the language of RCF while AlgIdx *is not with it.* Thus, our evaluation shall reveal the advantage of detecting and exploiting the invariance of the problem expressed in a language that directly encodes its geometric meaning.

Table 5. Percentage of Problems from which one or more Variables are Eliminated by the Rule for each Sort

GL_1	T_2	O_2	GL_2	T_3	O_3	GL_3	any
22.3	27.4	26.1	1.7	6.6	7.5	0.0	38.1

Table 6. Most Frequent Invariance Types Detected and Eliminated

Invariance	(%)	Invariance	(%)
GL_1, T_2, O_2	17.1	GL_1	2.4
T_2, O_2	8.1	GL_1, T_3, O_3	2.1
T_3, O_3	4.5	T_2, GL_2	1.1

Table 3 presents the results on all problems. The solver was run on each problem with a time limit of 600 s. The table lists the number of problems, the percentages of the problems solved within the time limit, and the average wall-clock time spent on the solved problems. The number of the solved problems is significantly increased in the **IMO** division. A modest improvement is observed in the other two divisions. Table 4 presents the results only on the problems in which at least one variable was eliminated by AlgIdx. The effect of the proposed method is quite clearly observed across all problem divisions and especially on **IMO**. On **IMO**, the average elapsed time on the problems solved by AlgIdx is longer than that by Baseline; it is because more difficult problems were solved by AlgIdx within the time limit. In fact, the average speed-up by AlgIdx (last column in Table 4) is around 500% on **Univ** and **Chart**; i.e., on the problems solved by both, AlgIdx output the answer five times faster than Baseline.

A curious fact is that both AlgIdx and Baseline tended to need more time to solve the problems on which an invariance was detected and eliminated by AlgIdx (i.e., Time in Table 4) than the average over all solved problems (Time in Table 3). It suggests that a problem having an invariance, or equivalently a symmetry, is harder for automatic solvers than those without it.

Figure 5 shows a comparison of the elapsed time for each problem. Each point represents a problem, and the x and y coordinates respectively indicate the elapsed time to solve (or to timeout) by Baseline and AlgIdx. We can see many problems that were not solved by Baseline within 600 s were solved within 300 s by AlgIdx. The speed-up is also observed on easier problems (those solved in 60 s) as can be seen in the right panel of Fig. 5.

Table 5 lists the fraction of problems on which one or more variables are eliminated based on the invariance indicated by an index variable of each sort. Table 6 provides the distribution of the combination of the sorts of invariances detected and eliminated by AlgIdx.

7 Conclusion

A method for automating w.l.o.g. arguments on geometry problems has been presented. It detects an invariance in a problem through type reconstruction in AIT and simplifies the problem utilizing the invariance. It was especially effective on harder problems including past IMO problems. Our future work includes the exploration for a more elaborate language of the index expressions that captures various kind of invariance while keeping the type inference amenable.

References

1. Aloul, F.A., Sakallah, K.A., Markov, I.L.: Efficient symmetry breaking for boolean satisfiability. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI 2003, pp. 271–276 (2003)
2. Arai, N.H.: Tractability of cut-free Gentzen type propositional calculus with permutation inference. Theoret. Comput. Sci. **170**(1), 129–144 (1996)
3. Arai, N.H., Urquhart, A.: Local symmetries in propositional logic. In: Dyckhoff, R. (ed.) TABLEAUX 2000. LNCS (LNAI), vol. 1847, pp. 40–51. Springer, Heidelberg (2000). https://doi.org/10.1007/10722086_3
4. Atkey, R., Johann, P., Kennedy, A.: Abstraction and invariance for algebraically indexed types. In: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2013, pp. 87–100 (2013)
5. Brown, C.W., Davenport, J.H.: The complexity of quantifier elimination and cylindrical algebraic decomposition. In: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation, ISSAC 2007, pp. 54–60 (2007)
6. Crawford, J.M., Ginsberg, M.L., Luks, E.M., Roy, A.: Symmetry-breaking predicates for search problems. In: Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning, KR 1996, pp. 148–159 (1996)
7. Davenport, J.H.: What does "without loss of generality" mean, and how do we detect it. Math. Comput. Sci. **11**(3), 297–303 (2017)
8. Devriendt, J., Bogaerts, B., Bruynooghe, M., Denecker, M.: Improved static symmetry breaking for SAT. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 104–122. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_8
9. Harrison, J.: Without loss of generality. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 43–59. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03359-9_3
10. Iwane, H., Anai, H.: Formula simplification for real quantifier elimination using geometric invariance. In: Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2017, pp. 213–220 (2017)
11. Kennedy, A.: Types for units-of-measure: theory and practice. In: Horváth, Z., Plasmeijer, R., Zsók, V. (eds.) CEFP 2009. LNCS, vol. 6299, pp. 268–305. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17685-2_8
12. Krishnamurthy, B.: Short proofs for tricky formulas. Acta Inform. **22**(3), 253–275 (1985)
13. Matsuzaki, T., Ito, T., Iwane, H., Anai, H., Arai, N.H.: Semantic parsing of pre-university math problems. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2017, pp. 2131–2141 (2017)
14. Matsuzaki, T., Iwane, H., Anai, H., Arai, N.H.: The most uncreative examinee: a first step toward wide coverage natural language math problem solving. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2014, pp. 1098–1104 (2014)

15. Matsuzaki, T., et al.: Can an A.I. win a medal in the mathematical olympiad? - Benchmarking mechanized mathematics on pre-university problems. AI Communications **31**(3), 251–266 (2018)
16. Matsuzaki, T., et al.: Race against the teens – benchmarking mechanized math on pre-university problems. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 213–227. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_15
17. Metin, H., Baarir, S., Colange, M., Kordon, F.: CDCLSym: introducing effective symmetry breaking in SAT solving. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10805, pp. 99–114. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89960-2_6
18. Reynolds, J.C.: Types, abstraction and parametric polymorphism. In: Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, pp. 513–523 (1983)
19. Sabharwal, A.: SymChaff: exploiting symmetry in a structure-aware satisfiability solver. Constraints **14**(4), 478–505 (2009)
20. Szeider, S.: Homomorphisms of conjunctive normal forms. Discrete Appl. Math. **130**(2), 351–365 (2003)
21. Wadler, P.: Theorems for free! In: Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture, FPCA 1989, pp. 347–359 (1989)

Synthetic Tableaux: Minimal Tableau Search Heuristics

Michał Sochański(✉), Dorota Leszczyńska-Jasion(✉), Szymon Chlebowski, Agata Tomczyk, and Marcin Jukiewicz

Adam Mickiewicz University, ul. Wieniawskiego 1, 61-712 Poznań, Poland
{Michal.Sochanski,Dorota.Leszczynska,Szymon.Chlebowski,
Agata.Tomczyk,Marcin.Jukiewicz}@amu.edu.pl

Abstract. We discuss the results of our work on heuristics for generating minimal synthetic tableaux. We present this proof method for classical propositional logic and its implementation in Haskell. Based on mathematical insights and exploratory data analysis we define heuristics that allows building a tableau of optimal or nearly optimal size. The proposed heuristics has been first tested on a data set with over 200,000 short formulas (length 12), then on 900 formulas of length 23. We describe the results of data analysis and examine some tendencies. We also confront our approach with the pigeonhole principle.

Keywords: Synthetic tableau · Minimal tableau · Data analysis · Proof-search heuristics · Haskell · Pigeonhole principle

1 Introduction

The method of *synthetic tableaux* (ST, for short) is a proof method based entirely on direct reasoning but yet designed in a tableau format. The basic idea is that all the laws of logic, and only laws of logic, can be derived directly by cases from parts of some partition of the whole logical space. Hence an ST-proof of a formula typically starts with a division between 'p-cases' and '$\neg p$-cases' and continues with further divisions, if necessary. Further process of derivation consists in applying the so-called *synthesizing* rules that build complex formulas from their parts—subformulas and/or their negations. For example, if p holds, then every implication with p in the succedent holds, '$q \to p$' in particular; then also '$p \to (q \to p)$' holds by the same argument. If $\neg p$ is the case, then every implication with p in the antecedent holds, thus '$p \to (q \to p)$' is settled. This kind of reasoning *proves* that '$p \to (q \to p)$' holds in every possible case (unless we reject *tertium non datur* in the partition of the logical space). There are no indirect assumptions, no *reductio ad absurdum*, no assumptions that need to be discharged. The ST method needs no labels, no derivation of a normal form (clausal form) is required.

This work was supported financially by National Science Centre, Poland, grant no 2017/26/E/HS1/00127.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 407–425, 2022.
https://doi.org/10.1007/978-3-031-10769-6_25

In the case of Classical Propositional Logic (CPL, for short) the method may be viewed as a formalization of the truth-tables method. The assumption that p amounts to considering all Boolean valuations that make p true; considering $\neg p$ exhausts the logical space. The number of cases to be considered corresponds to the number of branches of an ST, and it clearly depends on the number of distinct propositional variables in a formula, thus the upper bound for complexity of an ST-search is the complexity of the truth-tables method. In the worst case this is exponential with respect to the number of variables, but for some classes of formulas truth-tables behave better than standard analytic tableaux (see [4–7] for this diagnosis). However, the method of ST can perform better than truth-tables, as shown by the example of '$p \rightarrow (q \rightarrow p)$', where we do not need to partition the space of valuations against the $q/\neg q$ cases.[1] The question, obviously, is *how much* better? The considerations presented in this paper aim at developing a *quasi*-experimental framework for answering it.

The ST method was introduced in [19], then extended to some non-classical logics in [20,22]. An adjustment to the first-order level was presented in [14]. There were also interesting applications of the method in the domain of abduction: [12,13]. On the propositional level, the ST method is both a proof- and model-checking method, which means that one can examine satisfiability of a formula A (equivalently, validity of $\neg A$) and its falsifiability (equivalently, inconsistency of $\neg A$) at the same time. Normally, one needs to derive a clausal form of both A and $\neg A$ to check the two dual semantic cases (satisfiability and validity) with one of the quick methods, while the ST-system is designed to examine both of them. Wisely used, this property can contribute to limiting the increase in complexity in verification of semantic properties.

For the purpose of optimization of the ST method we created a heuristics that leads to construction of a variable ordering—a task similar to the one performed in research on Ordered Binary Decision Diagrams (OBDDs), and, generally, in Boolean satisfiability problem (SAT) [8,15]. In Sect. 3 we sketch a comparison of STs to OBDDs. Let us stress at this point, however, that the aim of our analysis remains proof-theoretical—the ST method is a 'full-blooded' proof method working on formulas of arbitrary representation. It was already adjusted to first-order and to some non-classical logics, and has a large scope of applications beyond satisfiability checking of clausal forms.

The optimization methods that we present are based on exploratory data analysis performed on millions of tableaux. Some aspects of the analysis are also discussed in the paper. The data are available on https://ddsuam.wordpress.com/software-and-data/.

Here is a plan of what follows. The next section introduces the ST method, Sect. 3 compares STs with analytic tableaux and with BDDs, and Sect. 4 presents the implementation in Haskell. In Sect. 5 we introduce the mathematical concepts

[1] On a side note, it is easy to show that the ST system is polynomially equivalent to system **KE** introduced in [4], as both systems contain cut. What is more, there is a strict analogy between the ST method and the inverse method (see [4,16]). The relation between ST and **KI** was examined by us in detail in Sect. 2 of [14].

needed to analyse heuristics of small tableaux generation. In Sect. 6 we describe the analysed data, and in Sect. 7—the obtained results. Section 8 confronts our approach with the pigeonhole principle, and Sect. 9 indicates plans for further research.

2 The Method of Synthetic Tableaux

Language. Let $\mathcal{L}_{\mathsf{CPL}}$ stand for the language of CPL with negation, $\neg$, and implication, $\rightarrow$. $\mathsf{Var} = \{p, q, r, \ldots, p_i, \ldots\}$ is the set of propositional variables and 'Form' stands for the set of all formulas of the language, where the notion of formula is understood in a standard way. $A, B, C \ldots$ will be used for formulas of $\mathcal{L}_{\mathsf{CPL}}$. Propositional variables and their negations are called *literals*. *Length* of a formula A is understood as the number of occurrences of characters in A, parentheses excluded.

Let $A \in \mathsf{Form}$. We define the notion of a *component* of A as follows. (i) A is a component of A. (ii) If A is of the form '$\neg\neg B$', then B is a component of A. (iii) If A is of the form '$B \rightarrow C$', then '$\neg B$' and C are components of A. (iv) If A is of the form '$\neg(B \rightarrow C)$', then B and '$\neg C$' are components of A. (v) If C is a component of B and B is a component of A, then C is a component of A. (vi) Nothing else is a component of A. By '$\mathsf{Comp}(A)$' we mean the set of all components of A. For example, $\mathsf{Comp}(\ p \rightarrow (q \rightarrow p)\ ) = \{p \rightarrow (q \rightarrow p), \neg p, q \rightarrow p, \neg q, p\}$. As we can see, *component of a formula* is not the same as *subformula of a formula*; $\neg q$ is not a subformula of the law of antecedent, q is, but it is not its component. Components refer to *uniform notation* as defined by Smullyan (see [18]) which is very convenient to use with a larger alphabet. Let us also observe that the association of $\mathsf{Comp}(A)$ with a Hintikka set is quite natural, although $\mathsf{Comp}(A)$ need not be consistent. In the sequel we shall also use '$\mathsf{Comp}^{\pm}(A)$' as a short for '$\mathsf{Comp}(A) \cup \mathsf{Comp}(\neg A)$'.

Rules. The system of ST consists of the set of rules (see Table 1) and the notion of proof (see Definition 2). The rules can be applied in the construction of an ST for a formula A on the proviso that (a) the premises already occur on a given branch, (b) the conclusion (conclusions, in the case of (*cut*)) of a particular application of the rule belongs (both belong) to $\mathsf{Comp}^{\pm}(A)$. The only branching rule, called (*cut*) by analogy to its famous sequent-calculus formulation, is at the same time the only rule that needs no premises, hence every ST starts with an application of this rule. If its application creates branches with p_i and $\neg p_i$, then we say that the rule was *applied with respect to p_i*.

One of the nice properties of this method is that it is easy to keep every branch *consistent*: it is sufficient to restrict the applications of (*cut*), so that on every branch (*cut*) is applied with respect to a given variable p_i at most once. This warrants that $p_i, \neg p_i$ never occur together on the same branch.

The notion of a proof is formalized by that of a tree. If $\mathcal{T}$ is a labelled tree, then by $X_{\mathcal{T}}$ we mean the set of its nodes, and by $r_{\mathcal{T}}$ we mean its root. Moreover, $\eta_{\mathcal{T}}$ is used for a function assigning labels to the nodes in $X_{\mathcal{T}}$.

Table 1. Rules of the ST system for $\mathcal{L}_{\mathsf{CPL}}$

$(\mathbf{r}^{\mathbf{1}}_{\rightarrow})$	$(\mathbf{r}^{\mathbf{2}}_{\rightarrow})$	$(\mathbf{r}^{\mathbf{3}}_{\rightarrow})$	$(\mathbf{r}_{\neg})$	(cut)
$\dfrac{\neg A}{A \rightarrow B}$	$\dfrac{B}{A \rightarrow B}$	$\dfrac{\begin{array}{c}A\\ \neg B\end{array}}{\neg(A \rightarrow B)}$	$\dfrac{A}{\neg\neg A}$	$p_i \qquad \neg p_i$

Definition 1 (synthetic tableau). *A* synthetic tableau *for a formula A is a finite labelled tree $\mathcal{T}$ generated by the above rules, such that $\eta_{\mathcal{T}} : X \backslash \{r_{\mathcal{T}}\} \longrightarrow \mathsf{Comp}^{\pm}(A)$ and each leaf is labelled with A or with $\neg A$.*

$\mathcal{T}$ is called consistent *if the applications of (cut) are subject to the restriction defined above: there are no two applications of (cut) on the same branch with respect to the same variable.*

$\mathcal{T}$ is called regular *provided that literals are introduced in the same order on each branch, otherwise $\mathcal{T}$ is called* irregular*.*

Finally, $\mathcal{T}$ is called canonical*, if, first, it is consistent and regular, and second, it starts with an introduction of all possible literals by (cut) and only after that the other rules are applied on the created branches.*

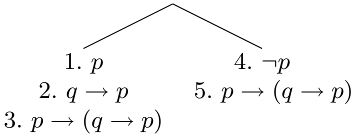
In the above definition we have used the notion of *literals introduced in the same order on each branch*. It seems sufficiently intuitive at the moment, so we postpone the clarification of this notion until the end of this section.

Definition 2 (proof in ST system). *A synthetic tableau $\mathcal{T}$ for a formula A is a* proof of A in the ST system *iff each leaf of $\mathcal{T}$ is labelled with A.*

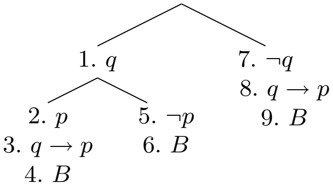
Theorem 1. (soundness and completeness, see [21]). *A formula A is valid in* CPL *iff A has a proof in the ST-system.*

Example 1. Below we present two different STs for one formula: $B = p \rightarrow (q \rightarrow p)$. Each of them is consistent and regular. Also, each of them is a proof of the formula in the ST system.

$\mathcal{T}_1$:


1. p
2. $q \rightarrow p$
3. $p \rightarrow (q \rightarrow p)$
4. $\neg p$
5. $p \rightarrow (q \rightarrow p)$

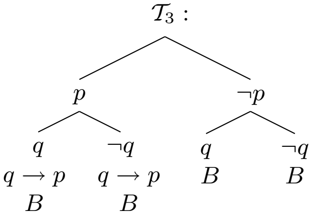
$\mathcal{T}_2$:

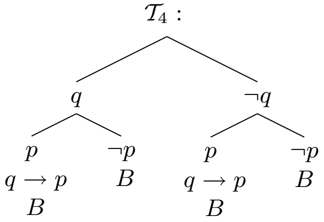


In $\mathcal{T}_1$: 2 comes from 1 by $\mathbf{r}^2_{\rightarrow}$, similarly 3 comes from 2 by $\mathbf{r}^2_{\rightarrow}$. 5 comes from 4 by $\mathbf{r}^1_{\rightarrow}$. In $\mathcal{T}_2$: nothing can be derived from 1, hence the application of (cut) wrt p is the only possible move. The numbering of the nodes is not part of the ST.

There are at least two important size measures used with respect to trees: the number of nodes and the number of branches. As witnessed by our data, there is a very high overall correlation between the two measures, we have thus used only one of them—the number of branches—in further analysis. Among various STs for the same formula there can be those of smaller, and those of bigger size. An ST of a minimal size is called *optimal*. In the above example, $\mathcal{T}_1$ is an optimal ST for B. Let us also observe that there can be many STs for a formula of the same size, in particular, there can be many optimal STs.

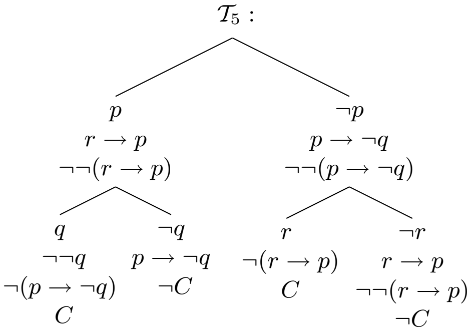
Example 2. Two possible canonical synthetic tableaux for $B = p \to (q \to p)$. Each of them is regular, consistent, but clearly not optimal (*cf.* $\mathcal{T}_1$).

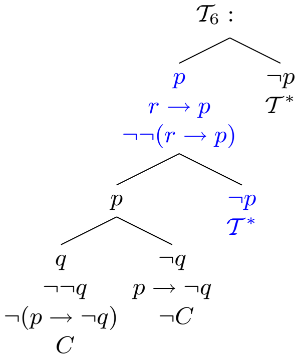




In the case of formulas with at most two distinct variables regularity is a trivial property. Here comes an example with three variables.

Example 3. $\mathcal{T}_5$ is an irregular ST for formula $C = (p \to \neg q) \to \neg(r \to p)$, i.e. variables are introduced in various orders on different branches. $\mathcal{T}_6$ is an example of an inconsistent ST for C, i.e. there are two applications of (*cut*) on one branch with respect to p, which results in a branch carrying both p and $\neg p$ (the blue one). The whole right subtree of $\mathcal{T}_5$, starting with $\neg p$, is repeated twice in $\mathcal{T}_6$, where it is symbolized with letter $\mathcal{T}^*$. Let us observe that $\neg\neg(r \to p)$ is a component of $\neg C$ due to clause (iv) defining the concept of component.





On the level of CPL we can use only consistent STs while still having a complete calculus (for details see [19,21]). An analogue of closing a branch of an analytic tableau for formula A is, in the case of an ST, ending a branch with

A synthesized. And the fact that an ST for A has a consistent branch ending with $\neg A$ witnesses satisfiability of $\neg A$. The situation concerning consistency of branches is slightly different, however, in the formalization of first-order logic presented in [14], as a restriction of the calculus to consistent STs produces an incomplete formalization.

Finally, let us introduce some auxiliary terminology to be used in the sequel. Suppose $\mathcal{T}$ is an ST for a formula A and $\mathcal{B}$ is a branch of $\mathcal{T}$. Literals occur on $\mathcal{B}$ in an order set by the applications of (*cut*), suppose that it is $\langle \pm p_1, \ldots, \pm p_n \rangle$, where '±' is a negation sign or no sign. In this situation we call sequence $o = \langle p_1, \ldots, p_n \rangle$ the *order on* $\mathcal{B}$. It can happen that o contains *all* variables that occur in A, or that some of them are missing. Suppose that $q_1, \ldots, q_m$ are all of (and only) the distinct variables occurring in A. Each permutation of $q_1, \ldots, q_m$ will be called *an instruction for a branch of an ST for* A. Further, we will say that the order o on $\mathcal{B}$ *complies with an instruction* I iff either $o = I$, or o constitutes a proper initial segment of I. Finally, $\mathcal{I}$ is an *instruction for the construction of* $\mathcal{T}$, if $\mathcal{I}$ is a set of instructions for branches of an ST for A such that for each branch of $\mathcal{T}$, the order on the branch complies with some element of $\mathcal{I}$.

Let us observe that in the case of a regular ST the set containing one instruction for a branch makes the whole instruction for the ST, as the instruction describes all the branches. Let us turn to examples. $\mathcal{T}_5$ from Example 3 has four branches with the following orders (from the left): $\langle p, q \rangle$, $\langle p, q \rangle$, $\langle p, r \rangle$, $\langle p, r \rangle$. On the other hand, there are six permutations of p, q, r, and hence six possible instructions for branches of an arbitrary ST for the discussed formula. Order $\langle p, q \rangle$ complies with instruction $\langle p, q, r \rangle$, and order $\langle p, r \rangle$ complies with instruction $\langle p, r, q \rangle$. The set $\{\langle p, q, r \rangle, \langle p, r, q \rangle\}$ is an instruction for the construction of an ST for C, more specifically, it is an instruction for the construction of $\mathcal{T}_5$.

3 ST, Analytic Tableaux, BDDs, and SAT Solvers

The analogy between STs and analytic tableaux sketched in the last paragraph of the previous section breaks in two points. First, let us repeat: the ST method is *both a satisfiability checker and a validity checker at once*, just like a truth table is. Second, the analogy breaks on complexity issues. In the case of analytic tableaux the order of decomposing compound formulas is the key to a minimal tableau. In the case of STs, the key to an optimized use of the method is a clever choice of variables introduced on each branch.

The main similarity between STs and Binary Decision Diagrams (BDDs, see e.g. [8,15]) is that both methods involve branching on variables. The main differences concern the representation they work on and their aims: firstly, STs constitute a proof method, whereas BDDs are compact representations of Boolean formulas, used mainly for practical aims such as design of electronic circuits (VLSI design); secondly, ST applies to logical formulas, whereas construction of BDDs may start with different representations of Boolean functions, usually circuits or Boolean formulas.

The structure of the constructed tree is also slightly different in the two approaches: in BDDs the inner nodes correspond to variables with outgoing edges labelled with 1 or 0; in STs, on the other hand, inner nodes are labelled with literals or more complex formulas. The terminal nodes of a BDD (also called sinks, labelled with 1 or 0) indicate the value of a Boolean function calculated for the arguments introduced along the path from the root, whereas the leaves of an ST carry a synthesized formula (the initial one or its negation). In addition to that, the methods differ in terms of the construction process: in case of BDDs, tree structures are first generated and then reduced to a more compact form using the elimination and merging rules; the STs, in turn, are built 'already reduced'. However, the interpretation of the outcome of both constructions is analogous. Firstly, for a formula A with n distinct variables $p_1, \ldots, p_n$ and the associated Boolean function $f_A = f_A(x_1, \ldots, x_n)$, the following fact holds: If a branch of an ST containing literals from a set L ends with A or $\neg A$ synthesized (which means that assuming that the literals from L are true is sufficient to calculate the value of A), then the two mentioned reduction rules can be used in a BDD for f_A, so that the route that contains the variables occurring in L followed by edges labelled according to the signs in L can be directed to a terminal node (sink). For example, if A can be synthesized on a branch with literals $\neg p_1$, p_2 and $\neg p_3$, then $f_A(0, 1, 0, x_4, \ldots, x_n) = 1$ for all values of the variables $y \in \{x_4, \ldots, x_n\}$ and so the route in the associated BDD containing the variables x_1, x_2 and x_3 followed by the edges labelled with 0, 1 and 0, respectively, leads directly to the sink labelled with 1.

However, possibility of applying the reduction procedures for a BDD does not always correspond to the possibility of reducing an ST. For example, the reduced BDD for formula $p \vee (q \wedge \neg q)$ consists of the single node labelled with p with two edges directed straight to the sinks 1 and 0; on the other hand, construction of an ST for the formula requires introducing q following the literal $\neg p$. This observation suggests that ST, in general, have greater size than the reduced BDDs.

Strong similarity of the two methods is also illustrated by the fact that they both allow the construction of a disjunctive normal form (DNF) of the logical or Boolean formula to which they were applied. In the case of ST, DNF is the disjunction of conjunctions of literals that appear on branches finished with the formula synthesized. The smaller the ST, the smaller the DNF. Things are analogous with BDDs.

Due to complexity issues, research on BDDs centers on ordered binary decision diagrams (OBDDs), in which different variables appear in the same order on all paths from the root. A number of heuristics have been proposed in order to construct a variable ordering that will lead to the smallest OBDDs, using characteristics of the different types of representation of Boolean function (for example, for circuits, topological characteristics have been used for that purpose). OBDDs are clearly analogous to regular STs, the construction of which also requires finding a good variable ordering, leading to a smaller ST. We suppose that our methodology can also be used to find orderings for OBDDs by expressing Boolean functions as logical formulas. It is not clear to us whether the OBDDs methodology can be used in our framework.

Let us move on to other comparisons, this time with a lesser degree of detail. It is very instructive to compare the ST method to SAT-solvers, as their effectiveness is undeniably impressive nowadays[2]. The ST method does not aim at challenging this effectiveness. Let us explain, however, in what aspect the ST method can still be viewed as a computationally attractive alternative to a SAT solver. The latter produces an answer to question about satisfiability, sometimes producing also examples of satisfying valuations and/or counting the satisfying valuations. In order to obtain an answer to another question—that about validity—one needs to ask about satisfiability of the initial problem negated. As we stressed above, the ST method answers the two questions at once, providing at the same time a description of classes of valuations satisfying and not satisfying the initial formula. Hence one ST is worth two SAT-checks together with a rough model counting.

Another interesting point concerns clausal forms. The method of ST does not require derivation of clausal form, but the applications of the rules of the system, defined via α-, β-notation, reflects the breaking of a formula into its components, and thus, in a way, leads to a definition of a normal form (a DNF, as we mentioned above). But this is not to say that an ST needs a full conversion to DNF. In this respect the ST method is rather similar to non-clausal theorem provers (*e.g.* non-clausal resolution, see [9,17]).

Let us finish this section with a summary of the ST method. Formally, it is a proof method with many applications beyond the realm of CPL. In the area of CPL, semantically speaking, it is both satisfiability and validity checker, displaying semantic properties of a formula like a truth table does, but amenable to work more efficiently (in terms of the number of branches) than the latter method. The key to this efficiency is in the order of variables introduced in an ST. In what follows we present a method of construction of such variable orders and examine our approach in an experimental setting.

4 Implementation

The main functionality of the implementation described in this section is a construction of an ST for a formula according to an instruction provided by the user. If required, it can also produce *all possible instructions* for a given formula and build all STs according to them. In our research we have mainly used the second possibility.

The implemented algorithm generates non-canonical, possibly irregular STs. Let us start with some basics. There are three main datatypes employed. Standard, recursively defined formula type, `For`, used to represent propositional formulas; Monad `Maybe Formula`, `MF`, consisting of `Just Formula` and `Nothing`—used to express the fact that the synthesis of a given formula on a given branch was successful (`Just`) or not (`Nothing`). To represent an ST we use type of trees imported from `Data.Tree`. Thus every ST can be represented as `Tree [MF]`

[2] See [23, p. 2021]: *contemporary SAT solvers can often handle practical instances with millions of variables and constraints.*

`[Tree [MF]]`, that is a tree labelled by lists of `MF`. We employed such a general structure having in mind possible extensions to non-classical logics (for CPL a binary tree is sufficient). The algorithm generating all possible ST for a given formula consists of the following steps:

1. We start by performing a few operations on the goal-formula A:
   (a) a list of all components of A and all components of $\neg A$, and a separate list of the variables occurring in A (`atoms A`) is generated;
   (b) the first list is sorted in such a way that all components of a given formula in that list precede it (`sort A`).
2. After this initial step, based on the list `atoms A`, all possible instructions for the construction of an ST for A are generated (`allRules (atoms A)`).
3. For each instruction from `allRules (atoms A)` we build an ST using the following strategy, called 'compulsory':
   (a) after each introduction of a literal (by (*cut*)) we try to synthesize (by the other rules) as many formulas from `sort A` as possible;
   (b) if no synthesizing rule is applicable we look into the instruction to introduce an appropriate literal and we go back to (a). Let us note that $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_5$ are constructed according to this strategy.
4. Lastly, we generate a CSV file containing some basic information about each generated tree: *int.al.* the number of nodes and whether the tree is a proof.

Please observe that the length of a single branch is linear in the size of a formula; this follows from the fact that `sort A` contains only the components of A. On the other hand, an 'outburst' of computational complexity enters on the level of the number of STs. In general, if k is the number of distinct variables in a formula A, then for $k = 3$ there are 12 different canonical STs, for $k = 4$ and $k = 5$ this number is, respectively, 576 and 1,688,800. In the case of $k = 6$ the number of canonical STs *per* formula exceeds 10^{12} and this approach is no longer feasible[3].

The Haskell implementation together with necessary documentation is available on https://ddsuam.wordpress.com/software-and-data/.

5 dp-Measure and the Rest of Our Toolbox

As we have already observed, in order to construct an optimal ST for a given formula one needs to make a clever choice of the literals to start with. The following function was defined to facilitate the smart choices. It assigns a rational value from the interval $\langle 0; 1\rangle$ to each occurrence of a literal in a syntactic tree for

[3] It can be shown (*e.g.* by mathematical induction) that for formulas with k different variables, the total number of canonical STs is given by the following explicit formula:

$$\prod_{i=1}^{k} (k-i+1)^{2^{i-1}}.$$

.

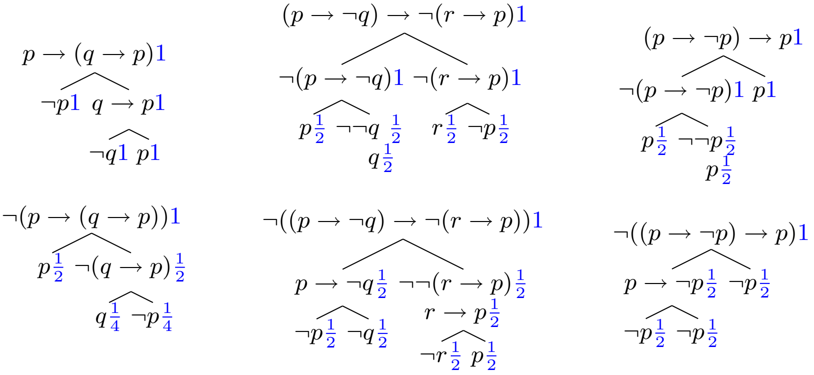
formula A (in fact, it assigns the values to all elements of $\mathsf{Comp}(A)$). Intuitively, the value reflects the *derivative power* of the literal in synthesizing A.

The first case of the equation in Definition 3 is to make the function full (=total) on $\mathsf{Form} \times \mathsf{Form}$, it also corresponds with the intended meaning of the defined measure: if $B \notin \mathsf{Comp}(A)$, then B is of no use in deriving A. The second case expresses the starting point: to calculate the values of $dp(A, B)$ for atomic B, one needs to assign $1 = dp(A, A)$; then the value is propagated down along the branches of a formula's syntactic tree. Dividing the value a by 2 in the fourth line reflects the fact that both components of an α-formula are needed to synthesize the formula. In order to use the measure, we need to calculate it for both A and $\neg A$; this follows from the fact that we do not know whether A or $\neg A$ will be synthesized on a given branch.

Definition 3. $dp : \mathsf{Form} \times \mathsf{Form} \longrightarrow \langle 0; 1\rangle$

$$dp(\ A, B\ ) = \begin{cases} 0 & \textit{if } B \notin \mathsf{Comp}(A), \\ 1 & \textit{if } B = A, \\ a & \textit{if } dp(\ A, \neg\neg B\ ) = a, \\ \frac{a}{2} & \textit{if } B \in \{C, \neg D\} \textit{ and } dp(\ A, \neg(C \rightarrow D)\ ) = a, \\ a & \textit{if } B \in \{\neg C, D\} \textit{ and } dp(\ A, C \rightarrow D\ ) = a. \end{cases}$$

Example 4. A visualization of calculating dp for formulas B, C from Examples 2, 3 and for $D = (p \rightarrow \neg p) \rightarrow p$.


$p \rightarrow (q \rightarrow p)1$
$\neg p1$ $q \rightarrow p1$
$\neg q1$ $p1$
$(p \rightarrow \neg q) \rightarrow \neg(r \rightarrow p)1$
$\neg(p \rightarrow \neg q)1$ $\neg(r \rightarrow p)1$
$p\frac{1}{2}$ $\neg\neg q\ \frac{1}{2}$ $r\frac{1}{2}$ $\neg p\frac{1}{2}$
$q\frac{1}{2}$
$(p \rightarrow \neg p) \rightarrow p1$
$\neg(p \rightarrow \neg p)1$ $p1$
$p\frac{1}{2}$ $\neg\neg p\frac{1}{2}$
$p\frac{1}{2}$
$\neg(p \rightarrow (q \rightarrow p))1$
$p\frac{1}{2}$ $\neg(q \rightarrow p)\frac{1}{2}$
$q\frac{1}{4}$ $\neg p\frac{1}{4}$
$\neg((p \rightarrow \neg q) \rightarrow \neg(r \rightarrow p))1$
$p \rightarrow \neg q\frac{1}{2}$ $\neg\neg(r \rightarrow p)\frac{1}{2}$
$\neg p\frac{1}{2}$ $\neg q\frac{1}{2}$ $r \rightarrow p\frac{1}{2}$
$\neg r\frac{1}{2}$ $p\frac{1}{2}$
$\neg((p \rightarrow \neg p) \rightarrow p)1$
$p \rightarrow \neg p\frac{1}{2}$ $\neg p\frac{1}{2}$
$\neg p\frac{1}{2}$ $\neg p\frac{1}{2}$

As one can see from Example 4, the effect of applying the dp measure to a formula and its negation is a number of values that need to be aggregated in order to obtain a clear instruction for an ST construction. However, some conclusions can be drawn already from the above example. It seems clear that the value $dp(\ p \rightarrow (q \rightarrow p), p\ ) = 1$ corresponds to the fact that p is sufficient to synthesize the whole formula (as witnessed by $\mathcal{T}_1$, see Example 1). So is the case with $\neg p$. On the other hand, even if $\neg q$ is sufficient to synthesize the formula, q is not (see $\mathcal{T}_2$, Example 1), hence the choice between p and q is plain. But it seems to be the only obvious choice at the moment. In the case of the second formula, every literal gets the same value: 0.5. What is more, in the case of longer formulas a situation depicted by the rightmost syntactic trees is very likely to happen:

we obtain $dp(D, p) = 0.5$ *twice* (since *dp* works on *occurrences* of literals), and $dp(\neg D, \neg p) = 0.5$ *three times.*

In the aggregation of the *dp*-values we use the parametrised Hamacher *s*-norm, defined for $a, b \in \langle 0; 1\rangle$ as follows:

$$a \,\mathbf{s}_\lambda\, b = \frac{a + b - ab - (1-\lambda)ab}{1 - (1-\lambda)ab}$$

for which we have taken $\lambda = 0.1$, as the value turned out to give the best results. Hamacher *s*-norm can be seen as a fuzzy alternative; it is commutative and associative, hence it is straightforward to extend its application to an arbitrary finite number of arguments. For $a = b = c = 0.5$ we obtain:

$$a \,\mathbf{s}_\lambda\, b \approx 0.677, \quad \text{and } (a \,\mathbf{s}_\lambda\, b) \,\mathbf{s}_\lambda\, c \approx 0.768$$

The value of this norm is calculated for a formula A and a literal l by taking the *dp*-values $dp(A, l)$ for each occurrence l in the syntactic tree of A. This value will be denoted as '$h(A, l)$'; in case there is only one value $dp(A, l)$, we take $h(A, l) = dp(A, l)$. Hence, referring to the above Example 4, we have *e.g.* $h(B, p) = 1$, $h(\neg B, \neg p) = 0.25$, $h(\neg D, \neg p) \approx 0.768$.

Finally, function H is defined for *variables*, not their occurrences, in formula A as follows:

$$H(A, p_i) = \frac{\max(h(A, p_i), h(\neg A, p_i)) + \max(h(A, \neg p_i), h(\neg A, \neg p_i))}{2}$$

The important property of this apparatus is that for $a, b < 1$ we have $a \,\mathbf{s}_{0.1}\, b > \max\{a, b\}$, and thus $h(A, l)$ and $H(A, p_i)$ are sensitive to the number of aggregated elements. Another desirable feature of the introduced functions is that $h(A, p_i) = 1$ indicates that one can synthesize A on a branch starting with p_i without further applications of (*cut*); furthermore, $H(A, p_i) = 1$ indicates that both p_i and $\neg p_i$ have this property.

Let us stress that the values of *dp*, *h* and *H* are very easy to calculate. Given a formula A, we need to assign a *dp*-value to each of its components, and the number of components is linear in the length of A. On the other hand, the information gained by these calculations is sometimes not sufficient. The assignment $dp(A, p_i) = 2^{-m}$ says only that A can be built from p_i and m other components of A, but it gives us no clue as to which components are needed. In Example 4, H works perfectly, as we have $H(B, p) = 1$ and $H(B, q) = 0.625$, hence H indicates the following instruction of construction of an ST: $\{\langle p, q\rangle\}$. Unfortunately, in the case of formula C we have $H(C, p) = H(C, q) = H(C, r) = 0.5$, hence a more sophisticated solution is needed.

6 Data

At the very beginning of the process of data generation we faced the following general problem: how to make any *conclusive* inferences about an infinite population (all Form) on the basis of finite data? Considering the methodological problems connected with applying classical statistical inference methods in this context, we limited our analysis to descriptive statistics, exploratory analysis and testing. To make this as informative as possible, we took a 'big data' approach: for every formula we generated all possible STs, differing in the order of applications of (*cut*) on particular branches. In addition to that, where it was feasible, we generated all possible formulas falling under some syntactical specifications. The approach is aimed at testing different optimisation methods as well as exploring data in search for patterns and new hypotheses. The knowledge gained in this way is further used on samples of longer formulas to examine tendencies.

From now on we use l for the length of a formula, k for the number of distinct variables occurring in a formula, and n for the number of all occurrences of variables (leaves, if we think of formulas as trees). On the first stage we examined a dataset containing all possible STs for formulas with $l = 12$ and $k \leqslant 4$. There are over 33 million of different STs already for these modest values; for larger k the data to analyse was simply too big. We generated 242,265 formulas, from which we have later removed those with $k \leqslant 2$ and/or $k = n$, as the results for them where not interesting. In the case of further datasets we also generated all possible STs, but the formulas were longer and they were randomly generated[4]. And so we considered (i) 400 formulas with $l = 23, k = 3$, (ii) 400 formulas with $l = 23, k = 4$, (iii) 100 formulas with $l = 23, k = 5$. In all cases $9 \leqslant n \leqslant 12$; this value is to be combined with the occurrences of negations in a formula—the smaller n, the more occurrences of negation.

Having all possible STs for a formula generated, we could simply check what is the optimal ST' size for this formula. The idea was to look for possible relations between, on the one hand, instructions producing the small STs, and, on the other hand, properties of formulas that are easy to calculate, like *dp* or numbers of occurrences of variables. The first dataset included only relatively small formulas; however, with all possible formulas of a given type available, it was possible *e.g.* to track various types of 'unusual' behaviour of formulas and all possible problematic issues regarding the optimisation methods, which could remain unnoticed if only random samples of formulas were generated. In case of randomly generated formulas the 'special' or 'difficult' types of formulas may not be tracked (as the probability of drawing them may be small), but instead we have an idea of an 'average' formula, or average behaviour of the optimisation methods. By generating all the STs, in turn, we gained access to full information not only about the regular but also irregular STs, which is the basis for indicating the set of optimal STs and the evaluation of the optimisation methods.

7 Data Analysis and a Discussion of Results

In this section we present some results of analyses performed on our data. The main purpose of the analyses is to test the effectiveness of the function H in terms

[4] The algorithm of generating random formulas is described in [11]. The author prepared also the Haskell implementation of the algorithm. See https://github.com/kiryk/random-for.

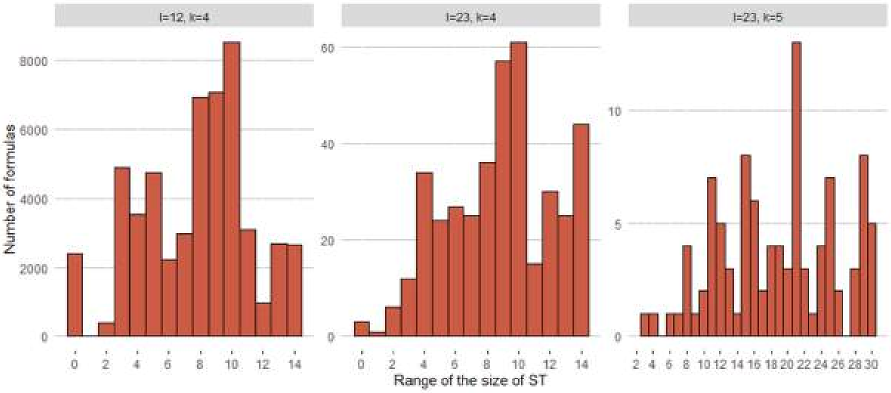


Fig. 1. Distribution of the difference between the size of a maximal and that of a minimal ST for formulas with $k = 4, 5$.

of indicating a small ST. Moreover, we performed different types of exploratory analysis on the data, aiming at understanding the variation of size among all STs for different formulas, and how it relates to the effectiveness of H.

Most results will be presented for the five combinations of the values of l and k in our data, that is, $l = 12, k \in \{3, 4\}$ and $l = 23, k \in \{3, 4, 5\}$; however, some results will be presented with the values of $k = 3$ and $k = 4$ grouped together (where the difference between them is insignificant) and the charts are presented only for $k \geqslant 4$.

We will examine the variation of size among STs using a range statistic: by *range of the size of ST for a formula A* (ST range, for short) we mean the difference between an ST of maximal and minimal size; this value indicates the possible room for optimization. The maximal-size ST is bounded by the size of a canonical ST for a given formula; its size depends only on k. For $k = 4$ a canonical ST has 16 branches, for $k = 5$ it is 32 branches.

The histograms on Fig. 1 present the distributions of ST range for formulas with $k = 4$ and $k = 5$. The rightmost bar in the histogram for $l = 23, k = 5$ says that for 5 (among 100) formulas there are STs with only two branches, where the maximal STs for these formulas have 32 branches. We can also read from the histograms that for formulas with $k = 4$ the ST range of some formulas is equal to 0 (7.9% of formulas with $l = 12$ and 3.5% with $l = 23$), which means that all STs have the same size. We have decided to exclude these formulas from the results of tests of efficiency of H, as the formulas leave no room for optimization. However, as can be seen on the histogram, there were no formulas of this kind among those with $k = 5$. This indicates that with the increase of k the internal differentiation of the set of STs for a formula increases as well, leading to a smaller share of formulas with small ST range.

Two more measures relating to the distribution of the size of ST may be of interest. Firstly, the share of formulas for which no regular ST is of optimal size—

Table 2. Row A: the share of formulas that do not have a regular ST of optimal size. Row B: the share of optimal STs among all STs for a formula; this was first calculated for each formula, then averaged over all formulas in a given set.

	$k = 3$		$k = 4$		$k = 5$
	$l = 12$	$l = 23$	$l = 12$	$l = 23$	$l = 23$
A	1.5%	1.1%	4.9%	3.3%	8.0%
B	31.7%	31.8%	17.3%	17.4%	10.0%

it indicates how wrong we can be in pointing to only the regular STs. Secondly, the percentage share of optimal STs among all STs for a given formula. The latter gives an idea what is the chance of picking an optimal ST at random. Table 2 presents both values for formulas depending on k and l (let us recall that formulas with ST range equal to 0 are excluded from the analysis). In both cases we can see clearly a tendency with growing k. As was to be expected, the table shows that the average share of optimal STs depends on the value of k rather than the size of the formula. This is understandable—as the number of branches depends on k only, the length of a formula translates to the length of branches, and the latter is linear in the former. In a way, this explains why the results are almost identical when the size of STs is calculated in terms of nodes rather than branches (as we mentioned above, the overall correlation between the two measures makes the choice between them irrelevant).

We can categorise the output of the function H into three main classes. In the first case, the values assigned to variables by H strictly order the variables, which results in one specific instruction of construction of a regular ST. The general score of such unique indications was very high: 70.9% for formulas with $l = 12$, 92.0% for $l = 23, k = 3, 4$, and 72.0% for $k = 5$. The second possibility is when H assigns the same value to each variable; in this case we gain no information at all (let us recall that we have excluded the only cases that could justify such assignments, that is, the formulas for which each ST is of the same size). The share of such formulas in our datasets was small: 0.6% for $l = 12$, 0.1% for $l = 23, k = 3, 4$ and 0% for $k = 5$, suggesting that it tends to fall with k rising. The third possibility is that the ordering is not strict, yet some information is gained. In this case for some, but not all, variables the value of H is the same.

The methodology used to asses effectiveness of H is quite simple. We assume that every indication must be a single regular instruction, hence we use additional criteria in case of formulas of the second and third kind described, in order to obtain a strict ordering. If H outputs the same value for some variables, we first order the variables by the number of occurrences in the formula; if the ordering is still not strict, we give priority to variables for which the sum of depths for all occurrences of literals in the syntactic tree is smaller; finally, where the above criteria do not provide a strict ordering, the order is chosen at random.

We used three evaluating functions to asses the quality of indications. Each function takes as arguments a formula and the ST for this formula indicated by

Table 3. The third column gives the number of formulas satisfying the characteristic presented in the first and the second column. The further three columns display values averaged on the sets. F_1 indicates how often we indicate an optimal ST. F_2 reports the mistake of our indication calculated as the difference of sizes between the indicated ST and an optimal one. Finally, POT indicates proximity to an optimal ST in a standardized way.

k	l	no of formulas	F_1	F_2	POT
3	12	113,190	0.935	0.089	0.974
	23	400	0.923	0.104	0.966
4	12	53,130	0.859	0.286	0.966
	23	400	0.836	0.297	0.960
5	23	100	0.75	0.52	0.971

our heuristics. The first function (F_1 in Table 3) outputs 1 if the indicated ST is of optimal size, 0 otherwise. The second function (F_2 in Table 3) outputs the difference between the size of the indicated ST and the optimal size. The third function is called *proximity to optimal tableau*, POT_A in symbols:

$$\mathsf{POT}_A(\mathcal{T}) = 1 - \frac{|\mathcal{T}| - min_A}{max_A - min_A}$$

where $\mathcal{T}$ is the ST for formula A indicated by H, $|\mathcal{T}|$ is the size of $\mathcal{T}$, max_A is the size of an ST for A of maximal size, and min_A is the size of an optimal ST for A. Later on we skip the relativization to A. Let us observe that the value $\frac{|\mathcal{T}|-min}{max-min}$ represents a mistake in indication relative to the ST range of a formula, and in this sense POT_A can be considered as a standardized measure of the quality of indication. Finally, values of each of the three evaluating functions were calculated for sets of formulas, by taking average values over all formulas in the set.

The results of the three functions presented in Table 3 show that optimal STs are indicated less often for formulas with greater k; however, the POT values seem to remain stable across all data, indicating that, on average, proximity of the indicated ST to the optimal ones does not depend on k or l.

Further analysis showed that the factor that most influenced the efficiency of our methodology was whether there is at least one value 1 among the *dp*-values of literals for a formula A. We shall write ‘Max(dp) $=$ 1’ if this is the case, and ‘Max(dp) $<$ 1’ otherwise (we skip the relativisation to A for simplicity). For formulas with Max(dp) $= 1$, results of the evaluating functions were much better; for example, the value of the POT function for formulas with $l = 12$ was 0.979 if Max(dp) $= 1$, and 0.814 for those with Max(dp) < 1; in case of formulas with $l = 23, k = 3, 4$ those values were 0.968 and 0.869, respectively, and for formulas with $l = 23, k = 5$ it was 0.974 and 0.901, respectively. This shows that our methodology works significantly worse if Max(dp) < 1; on the other hand, if Max(dp) $= 1$, the *dp* measure works very well. It should also be pointed

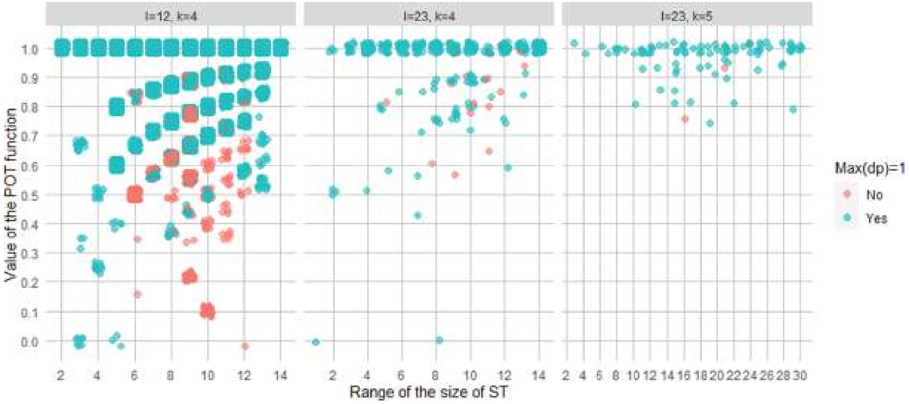


Fig. 2. Distribution of the difference between indicated and optimal ST in relation to ST-range. Every point corresponds to a formula, the points are slightly jittered in order to improve readibility. Each chart corresponds to different data, formulas $k = 3$ are excluded; additionally the colour indicates whether Max(dp) $= 1$ for a formula.

out that the difference between the POT values for both groups is smaller for formulas with greater l and k. Figure 2 presents a scatter plot that gives an idea of the whole distribution of the values of the POT function in relation to the ST range. Each formula on the plot is represented by a point, the colours additionally indicating whether Max(dp) < 1. The chart suggests, similarly as Table 3, that the method works well as the values of l and k rise for formulas, indicating STs that are on average equally close to the optimal ones.

One can point at two possible explanations of the fact that our methodology works worse for formulas with Max(dp) < 1. Firstly, if *e.g.*, $dp(A, p) = 2^{-m}$, we only obtain the information that, except for p, m more occurrences of components of A are required in order to synthesize the whole formula. Secondly, the function H neglects the complex dependencies between the various aggregated occurrences of a given variable, taking into account only the number of occurrences of literals in an aggregated group. However, considering very low computational complexity of the method based on the dp values and the function H, the outlined framework seems to provide good heuristics for indicating small STs. Methods that would reflect more aspects of the complex structure of logical formulas would likely require much more computational resources.

On a final note, we would like to add that exploration of the data allowed us to study properties of formulas that went beyond the scope of the optimisation of ST. The data was used in a similar way as in so called Experimental Mathematics, where numerous instances are analysed and visualized in order to *e.g.* gain insight, search for new patterns and relationships, test conjectures and introduce new concepts (see *e.g.* [1]).

Table 4. The pigeonhole principle

PHP_m			the size of ST		
m	k	l	indicated by H	minimal	canonical ST
1	2	7	3	3	4
2	6	34	15	11	64
3	12	90	99	43	4096
4	20	184	783	189	2^{20}

8 The Pigeonhole Principle

At the end we consider the propositional version of the principle introduced by Cook and Reckhow in [3, p. 43]. In the field of proof complexity the principle was used to prove that resolution is *intractable*, that is, any resolution proof of the propositional pigeonhole principle must be of exponential size (wrt the size of the formula). This has been proved by Haken in [10], see also[2].

Here is PHP_m in the propositional version:

$$\bigwedge_{0\leqslant i\leqslant m}\ \bigvee_{0\leqslant j<m} p_{i,j} \rightarrow \bigvee_{0\leqslant i<n\leqslant m}\ \bigvee_{0\leqslant j<m} (p_{i,j} \wedge p_{n,j})$$

where $\bigwedge$ and $\bigvee$ stand for generalized conjunction, disjunction (respectively) with the range indicated beneath.

The pigeonhole principle is constructed in a perfect symmetry of the roles played by the consecutive variables. Each variable has the same number of occurrences in the formula, and each of them gets the same value under H, they also have occurrences at the same depth of a syntactic tree. All this means that in our account we can only suggest a random, regular ST. However, it is worth noticing that, first, H behaves consistently with the structure of the formula, and second, the result is still attractive. In Table 4 the fourth column presents the size of the ST indicated by our heuristics, that is, in fact, generated by random ordering of variables. It is to be contrasted with the number 2^k in the last column describing the size of a canonical ST for the formula, which is at the same time the number of rows in a truth table for the formula. The minimal STs for the formulas were found with pen and paper and they are irregular.

9 Summary and Further Work

We presented a proof method of Synthetic Tableaux for CPL and explained how the efficiency of tableau construction depends on the choices of variables to apply (*cut*) to. We defined possible algorithms to choose the variables and experimentally tested their efficiency.

Our plan for the next research is well defined and it is to implement heuristics amenable to produce instructions for irregular STs. We have an algorithm, yet untested.

As far as proof-theoretical aims are concerned, the next task is to extend and adjust the framework to the first-order level based on the already described ST system for first-order logic [14]. We also wish to examine the efficiency of our indications on propositional non-classical logics for which the ST method exists (see [20,22]). In the area of data analysis another possible step would be to perform more complex statistical analysis using e.g. machine learning methods.

References

1. Borwein, J., Bailey, D.: Mathematics by Experiment: Plausible Reasoning in the 21st Century. A K Peters, Ltd., Natick (2004)
2. Buss, S.R.: Polynomial size proofs of the propositional pigeonhole principle. J. Symb. Log. **52**(4), 916–927 (1987)
3. Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. J. Symb. Log. **44**(1), 36–50 (1979)
4. D'Agostino, M.: Investigations into the complexity of some propositional calculi. Technical Monograph. Oxford University Computing Laboratory, Programming Research Group, November 1990
5. D'Agostino, M.: Are tableaux an improvement on truth-tables? Cut-free proofs and bivalence. J. Log. Lang. Comput. **1**, 235–252 (1992)
6. D'Agostino, M.: Tableau methods for classical propositional logic. In: D'Agostino, M., Gabbay, D.M., Hähnle, R., Posegga, J. (eds.) Handbook of Tableau Methods, pp. 45–123. Kluwer Academic Publishers (1999)
7. D'Agostino, M., Mondadori, M.: The taming of the cut. Classical refutations with analytic cut. J. Log. Comput. **4**(3), 285–319 (1994)
8. Ebendt, R., Fey, G., Drechsler, R.: Advanced BDD Opimization. Springer, Heidelberg (2005). https://doi.org/10.1007/b107399
9. Fitting, M.: First-Order Logic and Automated Theorem Proving, 2nd edn. Springer, New York (1996). https://doi.org/10.1007/978-1-4612-2360-3
10. Haken, A.: The intractability of resolution. Theoret. Comput. Sci. **39**, 297–308 (1985)
11. Kiryk, A.: A modified Korsh algorithm for random trees with various arity (manuscript) (2022)
12. Komosinski, M., Kups, A., Leszczyńska-Jasion, D., Urbański, M.: Identifying efficient abductive hypotheses using multi-criteria dominance relation. ACM Trans. Comput. Log. **15**(4), 1–20 (2014)
13. Komosinski, M., Kups, A., Urbański, M.: Multi-criteria evaluation of abductive hypotheses: towards efficient optimization in proof theory. In: Proceedings of the 18th International Conference on Soft Computing, Brno, Czech Republic, pp. 320–325 (2012)
14. Leszczyńska-Jasion, D., Chlebowski, S.: Synthetic tableaux with unrestricted cut for first-order theories. Axioms **8**(4), 133 (2019)
15. Meinel, C., Theobald, T.: Algorithms and Data Structures in VLSI Design. OBDD - Foundations and Applications, Springer, Heidelberg (1998). https://doi.org/10.1007/978-3-642-58940-9
16. Mondadori, M.: Efficient inverse tableaux. J. IGPL **3**(6), 939–953 (1995)
17. Murray, N.V.: Completely non-clausal theorem proving. Artif. Intell. **18**, 67–85 (1982)

18. Smullyan, R.M.: First-Order Logic. Springer, Berlin, Heidelberg, New York (1968). https://doi.org/10.1007/978-3-642-86718-7
19. Urbański, M.: Remarks on synthetic tableaux for classical propositional calculus. Bull. Sect. Log. **30**(4), 194–204 (2001)
20. Urbański, M.: Synthetic tableaux for Łukasiewicz' calculus Ł3. Logique Anal. (N.S.) **177–178**, 155–173 (2002)
21. Urbański, M.: Tabele syntetyczne a logika pytań (Synthetic Tableaux and the Logic of Questions). Wydawnictwo UMCS, Lublin (2002)
22. Urbański, M.: How to synthesize a paraconsistent negation. The case of CLuN. Logique Anal. **185–188**, 319–333 (2004)
23. Vizel, Y., Weissenbacher, G., Malik, S.: Boolean satisfiability solvers and their applications in model checking. Proc. IEEE **103**, 2021–2035 (2015)

Modal Logics

Paraconsistent Gödel Modal Logic

Marta Bílková[1], Sabine Frittella[2], and Daniil Kozhemiachenko[2](✉)

[1] The Czech Academy of Sciences, Institute of Computer Science,
Prague, Czech Republic
bilkova@cs.cas.cz

[2] INSA Centre Val de Loire, Univ. Orléans, LIFO EA 4022, Bourges, France
{sabine.frittella,daniil.kozhemiachenko}@insa-cvl.fr

Abstract. We introduce a paraconsistent modal logic $\mathbf{K}\mathsf{G}^2$, based on Gödel logic with coimplication (bi-Gödel logic) expanded with a De Morgan negation $\neg$. We use the logic to formalise reasoning with graded, incomplete and inconsistent information. Semantics of $\mathbf{K}\mathsf{G}^2$ is two-dimensional: we interpret $\mathbf{K}\mathsf{G}^2$ on crisp frames with two valuations v_1 and v_2, connected via $\neg$, that assign to each formula two values from the real-valued interval $[0, 1]$. The first (resp., second) valuation encodes the positive (resp., negative) information the state gives to a statement. We obtain that $\mathbf{K}\mathsf{G}^2$ is strictly more expressive than the classical modal logic $\mathbf{K}$ by proving that finitely branching frames are definable and by establishing a faithful embedding of $\mathbf{K}$ into $\mathbf{K}\mathsf{G}^2$. We also construct a constraint tableau calculus for $\mathbf{K}\mathsf{G}^2$ over finitely branching frames, establish its decidability and provide a complexity evaluation.

Keywords: Constraint tableaux · Gödel logic · Two-dimensional logics · Modal logics

1 Introduction

People believe in many things. Sometimes, they even have contradictory beliefs. Sometimes, they believe in one statement more than in the other. However, if a person has contradictory beliefs, they are not bound to believe in anything. Likewise, believing in ϕ *strictly more than* in χ makes one believe in ϕ *completely*. These properties of beliefs are natural, and yet hardly expressible in the classical modal logic. In this paper, we present a two-dimensional modal logic based on Gödel logic that can formalise beliefs taking these traits into account.

Two-Dimensional Treatment of Uncertainty. Belnap-Dunn four-valued logic (BD, or First Degree Entailment—FDE) [4,16,34] can be used to formalise

The research of Marta Bílková was supported by the grant 22-01137S of the Czech Science Foundation. The research of Sabine Frittella and Daniil Kozhemiachenko was funded by the grant ANR JCJC 2019, project PRELAP (ANR-19-CE48-0006). This research is part of the MOSAIC project financed by the European Union's Marie Skłodowska-Curie grant No. 101007627.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 429–448, 2022.
https://doi.org/10.1007/978-3-031-10769-6_26

reasoning with both incomplete and inconsistent information. In BD, formulas are evaluated on the De Morgan algebra $\mathbf{4}$ (Fig. 1, left) where the four values $\{t, f, b, n\}$ encode the information available about the formula: true, false, both true and false, neither true nor false. b and n thus represent inconsistent and incomplete information, respectively. It is important to note that the values represent the available information about the statement, not its intrinsic truth or falsity. Furthermore, this approach essentially treats *evidence for* a statement (its positive support) as being independent of *evidence against* it (negative support) which allows to differentiate between 'absence of evidence' and the 'evidence of absence'. The BD negation $\neg$ then swaps positive and negative supports.

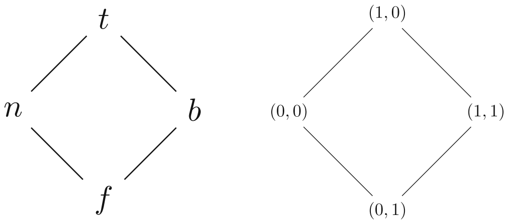


Fig. 1. $\mathbf{4}$ (left) and its continuous extension $[0,1]^{\bowtie}$ (right). $(x, y) \leq_{[0,1]^{\bowtie}} (x', y')$ iff $x \leq x'$ and $y \geq y'$.

The information regarding a statement, however, might itself be not crisp—after all, our sources are not always completely reliable. Thus, to capture the uncertainty, we extend $\mathbf{4}$ to the lattice $[0,1]^{\bowtie}$ (Fig. 1, right). $[0,1]^{\bowtie}$ is a twist product (cf, [37] for definitions) of $[0, 1]$ with itself: the order on the second coordinate is reversed w.r.t. the order on the first coordinate. This captures the intuition behind the usual 'truth' (upwards) order: an agent is more certain in χ than in ϕ when the evidence for χ is stronger than the evidence for ϕ while the evidence against χ is weaker than the evidence against ϕ.

Note that $[0,1]^{\bowtie}$ is a bilattice whose left-to-right order can be interpreted as the information order. This links the logics we consider to bilattice logics applied to reasoning in AI in [19] and then studied further in [24,35].

Comparing Beliefs. Uncertainty is manifested not only in the non-crisp character of the information. An agent might often lack the capacity to establish the concrete numerical value that represents their certainty in a given statement. Indeed, 'I am 43% certain that the wallet is Paula's' does not sound natural. On the other hand, it is reasonable to assume that the agents' beliefs can be compared in most contexts: neither 'I am more confident that the wallet is Paula's than that the wallet is Quentin's', nor 'Alice is more certain than Britney that Claire loves pistachio ice cream' require us to give a concrete numerical representation to the (un)certainty.

These considerations lead us to choosing the two-dimensional relative of the Gödel logic dubbed G^2 as the propositional fragment of our logic. G^2 was introduced in [5] and is, in fact, an extension of Moisil's logic[1] from [31] with the prelinearity axiom $(p \to q) \vee (q \to p)$. As in the original Gödel logic G, the validity of a formula in G^2 depends not on the values of its constituent variables but on the relative order between them. In this sense, G is a logic of comparative truth. Thus, as we treat positive and negative supports of a given statement independently, G^2 is a logic of comparative truth and falsity. Note that while the values of two statements may not be comparable (say, p is evaluated as $(0.5, 0.3)$ and q as $(0, 0)$), the coordinates of the values always are. We will see in Sect. 2, how we can formalise statements comparing agents' beliefs.

The sources available to the agents as well as the references between these sources can be represented as states in a Kripke model and its accessibility relation, respectively. It is important to mention that we account for the possibility that a source can give us contradictory information regarding some statement. Still, we want our reasoning with such information to be non-trivial. This is reflected by the fact that $(p \wedge \neg p) \to q$ is not valid in G^2. Thus, the logic (treated as a set of valid formulas) lacks the explosion principle. In this sense, we call G^2 and its modal expansions 'paraconsistent'. This links our approach to other paraconsistent fuzzy logics such as the ones discussed in [17].

To reason with the information provided by the sources, we introduce two interdefinable modalities—$\Box$ and $\Diamond$—interpreted as infima and suprema w.r.t. the upwards order on $[0, 1]^{\bowtie}$. We mostly assume (unless stated otherwise) that accessibility relations in models are crisp. Intuitively, it means that the sources are either accessible or not (and, likewise, either refer to the other ones, or not).

Broader Context. This paper is a part of the project introduced in [6] and carried on in [5] aiming to develop a modular logical framework for reasoning based on uncertain, incomplete and inconsistent information. We model agents who build their epistemic attitudes (like beliefs) based on information aggregated from multiple sources. $\Box$ and $\Diamond$ can be then viewed as two simple aggregation strategies: a pessimistic one (the infimum of positive support and the supremum of the negative support), and an optimistic one (the dual strategy), respectively. They can be defined via one another using $\neg$ in the expected manner: $\Box\phi$ stands for $\neg\Diamond\neg\phi$ and $\Diamond\phi$ for $\neg\Box\neg\phi$. In this paper, in contrast to [15] and [6], we do allow for modalities to nest.

The other part of our motivation comes from the work on modal Gödel logic ($\mathfrak{GK}$—in the notation of [36]) equipped with relational semantics [12,13, 36]. There, the authors develop proof and model theory of modal expansions of G interpreted over frames with both crisp and fuzzy accessibility relations. In particular, it was shown that the $\Box$-fragment[2] of $\mathfrak{GK}$ lacks the finite model property (FMP) w.r.t. fuzzy frames while the $\Diamond$-fragment has FMP[3] only w.r.t. fuzzy (but not crisp) frames. Furthermore, both $\Box$ and $\Diamond$ fragments of $\mathfrak{GK}$ are PSPACE-complete [28,29].

[1] This logic was introduced several times: by Wansing [38] as $\mathsf{I}_4\mathsf{C}_4$ and then by Leitgeb [27] as HYPE. Cf. [33] for a recent and more detailed discussion.

[2] Note that $\Box$ and $\Diamond$ are not interdefinable in $\mathfrak{GK}$—cf. [36, Lemma 6.1] for details.

[3] There is, however, a semantics in [11] w.r.t. which bi-modal $\mathfrak{GK}$ has FMP.

Description Gödel logics, a notational version of modal logics, have found their use the field of knowledge representation [8–10], in particular, in the representation of vague or uncertain data which is not possible in the classical ontologies. In this respect, our paper provides a further extension of representable data types as we model not only vague reasoning but also non-trivial reasoning with inconsistent information.

In the present paper, we are expanding the language with the Gödel coimplication $\prec$ to allow for the formalisation of statements expressing that an agent is *strictly more confident* in one statement than in another one (cf. Sect. 2 for the details). Furthermore, the presence of $\neg$ will allow us to simplify the frame definability. Still, we will show that our logic is a conservative extension of $\mathfrak{G}\mathfrak{K}^c$—the modal Gödel logic of crisp frames from [36] in the language with both $\Box$ and $\Diamond$.

Logics. We are discussing many logics obtained from the propositional Gödel logic G. Our main interest is in the logic we denote $\mathbf{K}\mathsf{G}^2$. It can be produced from G in several ways: (1) adding De Morgan negation $\neg$ to obtain G^2 (in which case $\phi \prec \phi'$ can be defined as $\neg(\neg\phi' \rightarrow \neg\phi)$) and then further expanding the language with $\Box$ or $\Diamond$; (2) adding $\prec$ or Δ (Baaz' delta) to G, then both $\Box$ and $\Diamond$ thus acquiring $\mathbf{K}\mathsf{biG}$[4] (modal bi-Gödel logic) which is further enriched with $\neg$. These and other relations are given on Fig. 2.

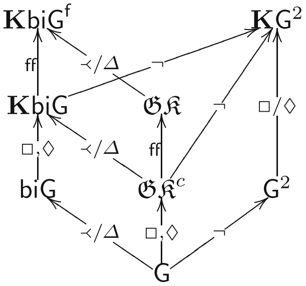


Fig. 2. Logics in the article. ff stands for 'permitting fuzzy frames'. Subscripts on arrows denote language expansions. / stands for 'or' and comma for 'and'.

Plan of the Paper. The remainder of the paper is structured as follows. In Sect. 2, we define bi-Gödel algebras and use them to present $\mathbf{K}\mathsf{biG}$ (on both fuzzy and crisp frames) and then $\mathbf{K}\mathsf{G}^2$ (on crisp frames), show how to formalise statements where beliefs of agents are compared, and prove some semantical properties. In Sect. 3, we show that $\Diamond$ fragment of $\mathbf{K}\mathsf{biG}^{\mathsf{f}}$ ($\mathbf{K}\mathsf{biG}$ on fuzzy frames) lacks finite model property. We then present a finitely branching fragment of $\mathbf{K}\mathsf{G}^2$ ($\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$) and argue for its use in representation of agents' beliefs. In Sect. 4, we design a constraint tableaux calculus for $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$ which we use to obtain the complexity results. Finally, in Sect. 5 we discuss further lines of research.

[4] To the best of our knowledge, the only work on bi-Gödel (symmetric Gödel) modal logic is [20]. There, the authors propose an expansion of biG with $\Box$ and $\Diamond$ equipped with proof-theoretic interpretation and provide its algebraic semantics.

2 Language and Semantics

In this section, we present semantics for $\mathbf{K}\mathsf{biG}$ (modal bi-Gödel logic) over both fuzzy and crisp frames and the one for $\mathbf{K}\mathsf{G}^2$ over crisp frames. Let Var be a countable set of propositional variables. The language $\mathsf{bi}\mathcal{L}^{\neg}_{\Box,\Diamond}$ is defined via the following grammar.

$$\phi := p \in \mathsf{Var} \mid \neg\phi \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\phi \prec \phi) \mid \Box\phi \mid \Diamond\phi$$

Two constants, $\mathbf{0}$ and $\mathbf{1}$, can be introduced in the traditional fashion: $\mathbf{0} := p \prec p$, $\mathbf{1} := p \rightarrow p$. Likewise, the Gödel negation can be also defined as expected: $\sim\phi := \phi \rightarrow \mathbf{0}$. The $\neg$-less fragment of $\mathsf{bi}\mathcal{L}^{\neg}_{\Box,\Diamond}$ is denoted with $\mathsf{bi}\mathcal{L}_{\Box,\Diamond}$.

To facilitate the presentation, we introduce bi-Gödel algebras.

Definition 1. *The bi-Gödel algebra* $[0,1]_{\mathsf{G}} = ([0,1], 0, 1, \wedge_{\mathsf{G}}, \vee_{\mathsf{G}}, \rightarrow_{\mathsf{G}}, \prec_{\mathsf{G}})$ *is defined as follows: for all* $a, b \in [0,1]$*, the standard operations are given by* $a \wedge_{\mathsf{G}} b := \min(a,b)$*,* $a \vee_{\mathsf{G}} b := \max(a,b)$*,*

$$a \rightarrow_G b = \begin{cases} 1, & \text{if } a \leq b \\ b & \text{else,} \end{cases} \qquad b \prec_G a = \begin{cases} 0, & \text{if } b \leq a \\ b & \text{else.} \end{cases}$$

Definition 2.

- *A* fuzzy frame *is a tuple* $\mathfrak{F} = \langle W, R \rangle$ *with* $W \neq \varnothing$ *and* $R : W \times W \rightarrow [0,1]$.
- *A* crisp frame *is a tuple* $\mathfrak{F} = \langle W, R \rangle$ *with* $W \neq \varnothing$ *and* $R \subseteq W \times W$.

Definition 3 ($\mathbf{K}\mathsf{biG}$ models). *A* $\mathbf{K}\mathsf{biG}$ model *is a tuple* $\mathfrak{M} = \langle W, R, v \rangle$ *with* $\langle W, R \rangle$ *being a (crisp or fuzzy) frame, and* $v : \mathsf{Var} \times W \rightarrow [0,1]$. v *(a valuation) is extended on complex* $\mathsf{bi}\mathcal{L}_{\Box,\Diamond}$ *formulas as follows:*

$$v(\phi \circ \phi', w) = v(\phi, w) \circ_{\mathsf{G}} v(\phi', w). \qquad (\circ \in \{\wedge, \vee, \rightarrow, \prec\})$$

The interpretation of modal formulas on fuzzy *frames is as follows:*

$$v(\Box\phi, w) = \inf_{w' \in W}\{wRw' \rightarrow_{\mathsf{G}} v(\phi, w')\}, \quad v(\Diamond\phi, w) = \sup_{w' \in W}\{wRw' \wedge_{\mathsf{G}} v(\phi, w')\}.$$

On crisp *frames, the interpretation is simpler (here,* $\inf(\varnothing) = 1$ *and* $\sup(\varnothing) = 0$*):*

$$v(\Box\phi, w) = \inf\{v(\phi, w') : wRw'\}, \qquad v(\Diamond\phi, w) = \sup\{v(\phi, w') : wRw'\}.$$

We say that $\phi \in \mathsf{bi}\mathcal{L}_{\Box,\Diamond}$ *is* $\mathbf{K}\mathsf{biG}$ valid on frame $\mathfrak{F}$ *(denote,* $\mathfrak{F} \models_{\mathbf{K}\mathsf{biG}} \phi$*) iff for any* $w \in \mathfrak{F}$*, it holds that* $v(\phi, w) = 1$ *for any model* $\mathfrak{M}$ *on* $\mathfrak{F}$.

Note that the definitions of validity in $\mathfrak{GK}^c$ and $\mathfrak{GK}$ coincide with those in $\mathbf{K}\mathsf{biG}$ and $\mathbf{K}\mathsf{biG}^{\mathsf{f}}$ if we consider the $\prec$-free fragment of $\mathsf{bi}\mathcal{L}_{\Box,\Diamond}$.

As we have already mentioned, on *crisp* frames, the accessibility relation can be understood as availability of (trusted or reliable) sources. In *fuzzy* frames, it can be thought of as the degree of trust one has in a source. Then, $\Diamond\phi$ represents the search for evidence from trusted sources that supports ϕ: $v(\Diamond\phi, t) > 0$ iff there is t' s.t. $tRt' > 0$ and $v(\phi, t') > 0$, i.e., there must be a source t' to which t has positive degree of trust and that has at least some certainty in ϕ. On the other hand, if no source is trusted by t (i.e., $tRu = 0$ for all u), then $v(\Diamond\phi, t) = 0$. Likewise, $\Box\chi$ can be construed as the search of evidence against χ given by trusted sources: $v(\Box\chi, t) < 1$ iff there is a source t' that gives to χ less certainty than t gives trust to t'. In other words, if t trusts no sources, or if all sources have at least as high confidence in χ as t has in them, then t fails to find a trustworthy enough counterexample.

Definition 4 (KG^2 models). *A* **KG**^{2} model *is a tuple* $\mathfrak{M} = \langle W, R, v_1, v_2\rangle$ *with* $\langle W, R\rangle$ *being a* crisp *frame, and* $v_1, v_2 : \mathsf{Var} \times W \to [0, 1]$*. The valuations which we interpret as support of truth and support of falsity, respectively, are extended on complex formulas as expected.*

$$\begin{array}{rlrl}
v_1(\neg\phi, w) &= v_2(\phi, w) & v_2(\neg\phi, w) &= v_1(\phi, w)\\
v_1(\phi \wedge \phi', w) &= v_1(\phi, w) \wedge_{\mathsf{G}} v_1(\phi', w) & v_2(\phi \wedge \phi', w) &= v_2(\phi, w) \vee_{\mathsf{G}} v_2(\phi', w)\\
v_1(\phi \vee \phi', w) &= v_1(\phi, w) \vee_{\mathsf{G}} v_1(\phi', w) & v_2(\phi \vee \phi', w) &= v_2(\phi, w) \wedge_{\mathsf{G}} v_2(\phi', w)\\
v_1(\phi \to \phi', w) &= v_1(\phi, w) \to_{\mathsf{G}} v_1(\phi', w) & v_2(\phi \to \phi', w) &= v_2(\phi', w) \prec_{\mathsf{G}} v_2(\phi, w)\\
v_1(\phi \prec \phi', w) &= v_1(\phi, w) \prec_{\mathsf{G}} v_1(\phi', w) & v_2(\phi \prec \phi', w) &= v_2(\phi', w) \to_{\mathsf{G}} v_2(\phi, w)\\
v_1(\Box\phi, w) &= \inf\{v_1(\phi, w') : wRw'\} & v_2(\Box\phi, w) &= \sup\{v_2(\phi, w') : wRw'\}\\
v_1(\Diamond\phi, w) &= \sup\{v_1(\phi, w') : wRw'\} & v_2(\Diamond\phi, w) &= \inf\{v_2(\phi, w') : wRw'\}
\end{array}$$

We say that $\phi \in \mathsf{bi}\mathcal{L}^{\neg}_{\Box,\Diamond}$ *is* **KG**^{2} valid on frame $\mathfrak{F}$ *(*$\mathfrak{F} \models_{\mathbf{KG}^2} \phi$*) iff for any* $w \in \mathfrak{F}$*, it holds that* $v_1(\phi, w) = 1$ *and* $v_2(\phi, w) = 0$ *for any model* $\mathfrak{M}$ *on* $\mathfrak{F}$*.*

Convention 1. *In what follows, we will denote a pair of valuations* $\langle v_1, v_2\rangle$ *just with* v *if there is no risk of confusion. Furthermore, for each frame* $\mathfrak{F}$ *and each* $w \in \mathfrak{F}$*, we denote*

$$\begin{aligned}
R(w) &= \{w' : wRw' = 1\}, && \text{(for fuzzy frames)}\\
R(w) &= \{w' : wRw'\}. && \text{(for crisp frames)}
\end{aligned}$$

Convention 2. *We will further denote with* **K**biG *the set of all formulas* **K**biG*-valid on all* crisp *frames;* **K**biG$^{\mathsf{f}}$ *the set of all formulas* **K**biG*-valid on all* fuzzy *frames; and* **KG**^{2}*—the set of all formulas* **KG**^{2} *valid on all* crisp *frames.*

Before proceeding to establish some semantical properties, let us make two remarks. First, neither $\Box$ nor $\Diamond$ are trivialised by contradictions: in contrast to **K**, $\Box(p \wedge \neg p) \to \Box q$ is not **KG**^{2} valid, and neither is $\Diamond(p \wedge \neg p) \to \Diamond q$. Intuitively, this means that one can have contradictory but non-trivial beliefs. Second, we can formalise statements of comparative belief such as the ones we have already given before:

> wallet: *I am more confident that the wallet is Paula's than that the wallet is Quentin's.*
> ice cream: *Alice is more certain than Britney that Claire loves pistachio ice cream.*

For this, consider the following defined operators.

$$\Delta\tau := {\sim}(\mathbf{1} \prec \tau) \tag{1}$$

$$\Delta^{\neg}\phi := {\sim}(\mathbf{1} \prec \phi) \wedge \neg{\sim}{\sim}(\mathbf{1} \prec \phi) \tag{2}$$

It is clear that for any $\tau \in \mathsf{bi}\mathcal{L}_{\Box,\Diamond}$ and $\phi \in \mathsf{bi}\mathcal{L}^{\neg}_{\Box,\Diamond}$ interpreted on $\mathbf{K}\mathsf{biG}$ and $\mathbf{K}\mathsf{G}^2$ models, respectively, it holds that

$$v(\Delta\tau, w) = \begin{cases} 1 & \text{if } v(\tau, w) = 1 \\ 0 & \text{otherwise,} \end{cases} \quad v(\Delta^{\neg}\phi, w) \quad = \begin{cases} (1,0) & \text{if } v(\phi, w) = (1,0) \\ (0,1) & \text{otherwise.} \end{cases} \tag{3}$$

Now we can define formulas that express order relations between values of two formulas both for $\mathbf{K}\mathsf{biG}$ and $\mathbf{K}\mathsf{G}^2$.

For $\mathbf{K}\mathsf{biG}$ they look as follows:

$$v(\tau, w) \leq v(\tau', w) \text{ iff } v(\Delta(\tau \to \tau'), w) = 1,$$
$$v(\tau, w) > v(\tau', w) \text{ iff } v\left({\sim}\Delta(\tau' \to \tau), w\right) = 1.$$

In $\mathbf{K}\mathsf{G}^2$, the orders are defined in a more complicated way:

$$v(\phi, w) \leq v(\phi', w) \text{ iff } v(\Delta^{\neg}(\phi \to \phi'), w) = (1,0),$$
$$v(\phi, w) > v(\phi', w) \text{ iff } v(\Delta^{\neg}(\phi' \to \phi) \wedge {\sim}\Delta^{\neg}(\phi \to \phi'), w) = (1,0).$$

Observe, first, that both in $\mathbf{K}\mathsf{biG}$ and $\mathbf{K}\mathsf{G}^2$ the relation 'the value of τ (ϕ) is less or equal to the value of τ' (ϕ')' is defined as '$\tau \to \tau'$ ($\phi \to \phi'$) has the designated value'. In $\mathbf{K}\mathsf{biG}$, the strict order is just a negation of the non-strict order since all values are comparable. On the other hand, in contrast to $\mathbf{K}\mathsf{biG}$, the strict order in $\mathbf{K}\mathsf{G}^2$ is not a simple negation of the non-strict order since $\mathbf{K}\mathsf{G}^2$ is essentially two-dimensional. We provide further details in Remark 2.

Finally, we can formalise wallet as follows. We interpret 'I am confident' as $\Box$ and substitute 'the wallet is Paula's' with p, and 'the wallet is Quentin's' with q. Now, we just use the definition of $>$ in $\mathsf{bi}\mathcal{L}^{\neg}_{\Box,\Diamond}$ to get

$$\Delta^{\neg}(\Box p \to \Box q) \wedge {\sim}\Delta^{\neg}(\Box q \to \Box p). \tag{4}$$

For ice cream, we need two different modalities: $\Box_a$ and $\Box_b$ for Alice and Brittney, respectively. Replacing 'Alice loves pistachio ice cream' with p, we get

$$\Delta^{\neg}(\Box_a p \to \Box_b p) \wedge {\sim}\Delta^{\neg}(\Box_b p \to \Box_a p). \tag{5}$$

Remark 1. Δ is called Baaz' delta (cf., e.g. [3] for more details). Intuitively, $\Delta\tau$ can be interpreted as 'τ has the designated value' and acts much like a necessity modality: if τ is $\mathbf{K}\mathsf{biG}$ valid, then so is $\Delta\tau$; moreover, $\Delta(p \to q) \to (\Delta p \to \Delta q)$ is valid. Furthermore, Δ and $\prec$ can be defined via one another in $\mathbf{K}\mathsf{biG}$, thus the addition of Δ to G makes it more expressive and allows to define both strict and non-strict orders.

Remark 2. Recall that we mentioned in Sect. 1 that an agent should usually be able to compare their beliefs in different statements: this is reflected by the fact that $\Delta(p \rightarrow q) \vee \Delta(q \rightarrow p)$ is $\mathbf{K}\mathsf{biG}$ valid. It can be counter-intuitive if the contents of beliefs have nothing in common, however.

This drawback is avoided if we treat support of truth and support of falsity independently. Here is where a difference between $\mathbf{K}\mathsf{biG}$ and $\mathbf{K}\mathsf{G}^2$ lies. In $\mathbf{K}\mathsf{G}^2$, we can *only compare the values of formulas coordinate-wise*, whence $\Delta^{\neg}(p \rightarrow q) \vee \Delta^{\neg}(q \rightarrow p)$ is not $\mathbf{K}\mathsf{G}^2$ valid. E.g., if we set $v(p,w) = (0.7, 0.6)$ and $v(q,w) = (0.4, 0.2)$, $v(p,w)$ and $v(q,w)$ will not be comparable w.r.t. the truth (upward) order on $[0,1]^{\bowtie}$.

We end this section with establishing some useful semantical properties.

Proposition 1. *$\mathfrak{F} \models_{\mathbf{K}\mathsf{G}^2} \phi$ iff for any model $\mathfrak{M}$ on $\mathfrak{F}$ and any $w \in \mathfrak{F}$, $v_1(\phi, w) = 1$.*

Proof. The 'if' direction is evident from the definition of validity. We show the 'only if' part. It suffices to show that the following statement holds for any ϕ and $w \in \mathfrak{F}$:

> *for any $v(p,w) = (x,y)$, let $v^*(p,w) = (1-y, 1-x)$. Then $v(\phi,w) = (x,y)$ iff $v^*(\phi,w) = (1-y, 1-x)$.*

We proceed by induction on ϕ. The proof of propositional cases is identical to the one in [5, Proposition 5]. We consider only the case of $\phi = \Box\psi$ since $\Box$ and $\Diamond$ are interdefinable.

Let $v(\Box\psi, w) = (x,y)$. Then $\inf\{v_1(\psi, w') : wRw'\} = x$, and $\sup\{v_2(\psi, w') : wRw'\} = y$. Now, we apply the induction hypothesis to ψ, and thus if $v(\psi, s) = (x', y')$, then $v^*(\psi, s) = (1-y', 1-x')$ for any $s \in R(w)$. But then $\inf\{v_1^*(\psi, w') : wRw'\} = 1 - y$, and $\sup\{v_2^*(\psi, w') : wRw'\} = 1 - x$ as required.

Now, assume that $v_1(\phi, w) = 1$ for any v_1 and w. We can show that $v_2(\phi, w) = 0$ for any w and v_2. Assume for contradiction that $v_2(\phi, w) = y > 0$ but $v_1(\phi, w) = 1$. Then, $v^*(\phi) = (1-y, 1-1) = (1-y, 0)$. But since $y > 0$, $v^*(\phi) \neq (1, 0)$.

Proposition 2.

1. *Let ϕ be a formula over $\{\mathbf{0}, \wedge, \vee, \rightarrow, \Box, \Diamond\}$. Then, $\mathfrak{F} \models_{\mathfrak{G}\mathfrak{K}} \phi$ iff $\mathfrak{F} \models_{\mathbf{K}\mathsf{biG}^{\mathsf{f}}} \phi$ and $\mathfrak{F} \models_{\mathfrak{G}\mathfrak{K}^c} \phi$ iff $\mathfrak{F} \models_{\mathbf{K}\mathsf{biG}} \phi$, for any $\mathfrak{F}$.*
2. *Let $\phi \in \mathsf{bi}\mathcal{L}_{\Box,\Diamond}$. Then, $\mathfrak{F} \models_{\mathbf{K}\mathsf{biG}} \phi$ iff $\mathfrak{F} \models_{\mathbf{K}\mathsf{G}^2} \phi$, for any crisp $\mathfrak{F}$.*

Proof. 1. follows directly from the semantic conditions of Definition 3. We consider 2. The 'only if' direction is straightforward since the semantic conditions of v_1 in $\mathbf{K}\mathsf{G}^2$ models and v in $\mathbf{K}\mathsf{biG}$ models coincide. The 'if' direction follows from Proposition 1: if ϕ is valid on $\mathfrak{F}$, then $v(\phi, w) = 1$ for any $w \in \mathfrak{F}$ and any v on $\mathfrak{F}$. But then, $v_1(\phi, w) = 1$ for any $w \in \mathfrak{F}$. Hence, $\mathfrak{F} \models_{\mathbf{K}\mathsf{G}^2} \phi$.

3 Model-Theoretic Properties of $\mathbf{K}\mathsf{G}^2$

In the previous section, we have seen how the addition of $\prec$ allowed us to formalise statements considering comparison of beliefs. Here, we will show that both $\Box$ and $\Diamond$ fragments of $\mathbf{K}\mathsf{biG}$, and hence $\mathbf{K}\mathsf{G}^2$, are strictly more expressive than the classical modal logic $\mathbf{K}$, i.e. that they can define all classically definable classes of crisp frames as well as some undefinable ones.

Definition 5 (Frame definability). *Let Σ be a set of formulas. Σ* defines *a class of frames $\mathbb{K}$ in a logic $\mathbf{L}$ iff it holds that $\mathfrak{F} \in \mathbb{K}$ iff $\mathfrak{F} \models_{\mathbf{L}} \Sigma$.*

The next statement follows from Proposition 2 since $\mathbf{K}$ can be faithfully embedded in $\mathfrak{GK}^c$ by substituting each variable p with $\sim\sim p$ (cf. [28,29] for details).

Theorem 1. *Let $\mathbb{K}$ be a class of frames definable in $\mathbf{K}$. Then, $\mathbb{K}$ is definable in $\mathbf{K}\mathsf{biG}$ and $\mathbf{K}\mathsf{G}^2$.*

Theorem 2. *1. Let $\mathfrak{F}$ be* crisp*. Then $\mathfrak{F}$ is finitely branching (i.e., $R(w)$ is finite for every $w \in \mathfrak{F}$) iff $\mathfrak{F} \models_{\mathbf{K}\mathsf{biG}} \mathbf{1} \prec \Diamond((p \prec q) \wedge q)$.*
2. Let $\mathfrak{F}$ be fuzzy*. Then $\mathfrak{F}$ is finitely branching and $\sup\{wRw' : wRw' < 1\} < 1$ for all $w \in \mathfrak{F}$ iff $\mathfrak{F} \models_{\mathbf{K}\mathsf{biG}} \mathbf{1} \prec \Diamond((p \prec q) \wedge q)$.*

Proof. We show the case of fuzzy frames since the crisp ones can be tackled in the same manner. Assume that $\mathfrak{F}$ is finitely branching and that $\sup\{wRw' : wRw' < 1\} < 1$ for all $w \in \mathfrak{F}$. It suffices to show that $v(\Diamond((p \prec q) \wedge q), w) < 1$ for all $w \in \mathfrak{F}$. First of all, observe that there is no $w' \in \mathfrak{F}$ s.t. $v((p \prec q) \wedge q, w') = 1$. It is clear that $\sup\limits_{wRw' < 1} \{v((p \prec q) \wedge q, w') \wedge_{\mathsf{G}} wRw'\} < 1$ and that

$$\sup\{v((p \prec q) \wedge q, w') : wRw' = 1\} = \max\{v((p \prec q) \wedge q, w') : wRw' = 1\} < 1$$

since $R(w)$ is finite. But then $v(\Diamond((p \prec q) \wedge q), w) < 1$ as required.

For the converse, either (1) $R(w)$ is infinite for some w, or (2) $\sup\{wRw' : wRw' < 1\} = 1$ for some w. For (1), set $v(p, w') = 1$ for every $w' \in R(w)$. Now let $W' \subseteq R(w)$ and $W' = \{w_i : i \in \{1, 2, \ldots\}\}$. We set $v(q, w_i) = \frac{i}{i+1}$. It is easy to see that $\sup\{v(q, w_i) : w_i \in W'\} = 1$ and that $v((p \prec q) \wedge q, w_i) = v(q, w_i)$. Therefore, $v(\mathbf{1} \prec \Diamond((p \prec q) \wedge q), w) = 0$.

For (2), we let $v(p, w') = 1$ and further, $v(q, w') = wRw'$ for all $w' \in \mathfrak{F}$. Now since $\sup\{wRw' : wRw' < 1\} = 1$ and $v(((p \prec q) \wedge q), w') = v(q, w')$ for all $w' \in \mathfrak{F}$, it follows that $v(\Diamond((p \prec q) \wedge q), w) = 1$, whence $v(\mathbf{1} \prec \Diamond((p \prec q) \wedge q), w) = 0$.

Remark 3. The obvious corollary of Theorem 2 is the lack of FMP for the $\Diamond$-fragment of $\mathbf{K}\mathsf{biG}^{\mathsf{f}}$[5] since $\Diamond((p \prec q) \wedge q)$ in never true in a finite model. This differentiates $\mathbf{K}\mathsf{biG}^{\mathsf{f}}$ from $\mathfrak{GK}$ since the $\Diamond$-fragment of $\mathfrak{GK}$ *has* FMP [12, Theorem 7.1]. Moreover, one can define finitely branching frames in $\Box$ fragments of $\mathfrak{GK}$ and $\mathfrak{GK}^c$. Indeed, $\sim\sim\Box(p \vee \sim p)$ serves as such definition.

Corollary 1. *$\mathbf{K}\mathsf{G}^2$ and both $\Box$ and $\Diamond$ fragments of $\mathbf{K}\mathsf{biG}$ are strictly more expressive than $\mathbf{K}$.*

Proof. From Theorems 1 and 2 since $\mathbf{K}$ is complete both w.r.t. all frames and all finitely branching frames. The result for $\mathbf{K}\mathsf{G}^2$ follows since it is conservative over $\mathbf{K}\mathsf{biG}$ (Proposition 2).

[5] Bi-modal $\mathbf{K}\mathsf{biG}^{\mathsf{f}}$ lacks have FMP since it is a conservative extension of $\mathfrak{GK}$.

These results show us that addition of $\prec$ greatly enhances the expressive power of our logic. Here it is instructive to remind ourselves that classical epistemic logics are usually complete w.r.t. finitely branching frames (cf. [18] for details). It is reasonable since for practical reasoning, agents cannot consider infinitely many alternatives. In our case, however, if we wish to use $\mathbf{K}\mathsf{biG}$ and $\mathbf{K}\mathsf{G}^2$ for knowledge representation, we need to *impose* finite branching explicitly.

Furthermore, allowing for infinitely branching frames in $\mathbf{K}\mathsf{biG}$ or $\mathbf{K}\mathsf{G}^2$ leads to counter-intuitive consequences. In particular, it is possible that $v(\Box\phi, w) = (0, 1)$ even though there are no $w', w'' \in R(w)$ s.t. $v_1(\phi, w') = 0$ or $v_2(\phi, w'') = 1$. In other words, there is no source that decisively falsifies ϕ, furthermore, all sources have some evidence *for* ϕ, and yet we somehow believe that ϕ is completely false and untrue. Dually, it is possible that $v(\Diamond\phi, w) = (1, 0)$ although there are no $w', w'' \in R(w)$ s.t. $v_1(\phi, w') = 1$ or $v_2(\phi, w'') = 0$. Even though $\Diamond$ is an 'optimistic' aggregation, it should not ignore the fact that *all* sources have some evidence *against* ϕ but *none* supports it completely.

Of course, this situation is impossible if we consider only finitely branching frames for infima and suprema will become minima and maxima. There, all values of modal formulas will be *witnessed* by some accessible states in the following sense. For $\heartsuit \in \{\Box, \Diamond\}$, $i \in \{1, 2\}$, if $v_i(\heartsuit\phi, w) = x$, then there is $w' \in R(w)$ s.t. $v_i(\phi, w') = x$. Intuitively speaking, finitely branching frames represent the situation when our degree of certainty in some statement is based uniquely on the data given by the sources.

Convention 3. *We will further use* $\mathbf{K}\mathsf{biG}_{\mathsf{fb}}$ *and* $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$ *to denote the sets of all* $\mathsf{bi}\mathcal{L}_{\Box,\Diamond}$ *and* $\mathsf{bi}\mathcal{L}^{\neg}_{\Box,\Diamond}$ *formulas valid on finitely branching crisp frames.*

Observe, moreover, that $\Box$ and $\Diamond$ are still undefinable via one another in $\mathsf{bi}\mathcal{L}_{\Box,\Diamond}$. The proof is the same as that of [36, Lemma 6.1].

Proposition 3. $\Box$ *and* $\Diamond$ *are not interdefinable in* $\mathbf{K}\mathsf{biG}_{\mathsf{fb}}$.

Corollary 2.

1. $\Box$ *and* $\Diamond$ *are not interdefinable in* $\mathbf{K}\mathsf{biG}$, $\mathbf{K}\mathsf{biG}^{\mathsf{f}}_{\mathsf{fb}}$, *and* $\mathbf{K}\mathsf{biG}^{\mathsf{f}}$.
2. *Both* $\Box$ *and* $\Diamond$ *fragments of* $\mathbf{K}\mathsf{biG}$ *are more expressive than* $\mathbf{K}$.

In the remainder of the paper, we are going to provide a complete proof system for $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$ (and hence, $\mathbf{K}\mathsf{biG}_{\mathsf{fb}}$), and establish its decidability and complexity as well as finite model property. Note, however, that the latter is not entirely for granted. In fact, several expected ways of defining filtration (cf. [7,14] for more details thereon) fail.

Let $\Sigma \subseteq \mathsf{bi}\mathcal{L}_{\Box,\Diamond}$ be closed under subformulas. If we want to have filtration for $\mathbf{K}\mathsf{biG}_{\mathsf{fb}}$, there are three intuitive ways to define $\sim_\Sigma$ on the carrier of a model that is supposed to relate states satisfying the same formulas.

1. $w \sim^1_\Sigma w'$ iff $v(\phi, w) = v(\phi, w')$ for all $\phi \in \Sigma$.
2. $w \sim^2_\Sigma w'$ iff $v(\phi, w) = 1 \Leftrightarrow v(\phi, w') = 1$ for all $\phi \in \Sigma$.
3. $w \sim^3_\Sigma w'$ iff $v(\phi, w) \leq v(\phi', w) \Leftrightarrow v(\phi, w') \leq v(\phi', w')$ for all $\phi, \phi' \in \Sigma \cup \{\mathbf{0}, \mathbf{1}\}$.

Consider the model on Fig. 3 and two formulas:

$$\phi^{\leq} := {\sim}{\sim}(p \rightarrow \lozenge p) \qquad\qquad \phi^{>} := {\sim}{\sim}(p \prec \lozenge p)$$

Now let Σ to be the set of all subformulas of $\phi^{\leq} \wedge \phi^{>}$.

First of all, it is clear that $v(\phi^{\leq} \wedge \phi^{>}, w) = 1$ for any $w \in \mathfrak{M}$. Observe now that all states in $\mathfrak{M}$ are *distinct* w.r.t. $\sim^1_\Sigma$. Thus, the first way of constructing the carrier of the new model does not give the FMP.

$$\mathfrak{M} : w_1 \longrightarrow w_2 \longrightarrow \ldots \longrightarrow w_n \longrightarrow \ldots$$

Fig. 3. $v(p, w_n) = \frac{1}{n+1}$

As regards to $\sim^2_\Sigma$ and $\sim^3_\Sigma$, one can check that for any $w, w' \in \mathfrak{M}$, it holds that $w \sim^2_\Sigma w'$ and $w \sim^3_\Sigma w'$. So, if we construct a filtration of $\mathfrak{M}$ using equivalence classes of either of these two relations, the carrier of the resulting model is going to be finite. Even more so, it is going to be a singleton.

However, we can show that there is *no finite model* $\mathfrak{N} = \langle U, S, e \rangle$ s.t.

$$\forall s \in \mathfrak{N} : v(\phi^{\leq} \wedge \phi^{>}, s) = 1.$$

Indeed, $e(\phi^{\leq}, t) = 1$ iff $e(p, t') > 0$ for some $t' \in S(t)$, while $e(\phi^{>}, t) = 1$ iff $v(p, t) > v(p, t')$ for any $t' \in S(t)$. Now, if U is finite, we have two options: either (1) there is $u \in U$ s.t. $R(u) = \varnothing$, or (2) U contains a finite S-cycle.

For (1), note that $v(\lozenge p, u) = 0$, and we have two options: if $e(p, u) = 0$, then $e(\phi^{>}, u) = 0$; if, on the other hand, $e(p, u) > 0$, then $e(\phi^{\leq}, u) = 0$. For (2), assume w.l.o.g. that the S-cycle looks as follows: $u_0 S u_1 S u_2 \ldots S u_n S u_0$.

If $e(p, u_0) = 0$, $e(\phi^{>}, u_0) = 0$, so $e(p, u_0) > 0$. Furthermore, $e(p, u_i) > e(p, u_{i+1})$. Otherwise, again, $e(\phi^{>}, u_i) = 0$. But then we have $e(\phi^{>}, u_i) = 0$.

But this means that $\sim^2_\Sigma$ and $\sim^3_\Sigma$ do not preserve truth of formulas from w to $[w]_\Sigma$, i.e., neither of these two relations can be used to define filtration. Thus, in order to explicitly prove the finite model property and establish complexity evaluations for $\mathbf{K}\mathsf{biG}_{\mathsf{fb}}$ and $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$, we will provide a tableaux calculus. It will also serve as a decision procedure for satisfiability and validity of formulas.

4 Tableaux for $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$

Usually, proof theory for modal and many-valued logics is presented in one of the following several forms. The first one is a Hilbert-style axiomatisation as given in e.g. [23] for the propositional Gödel logic and in [12,13,36] for its modal expansions. Hilbert calculi are useful for establishing frame correspondence results as well as for showing that one logic extends another one in the same language. On the other hand, their completeness proofs might be quite complicated, and the proof-search not at all straightforward. Second, there are non-labelled sequent and hyper-sequent calculi (cf. [30] for the propositional proof systems and [28,29] for the modal hypersequent calculi). With regards to modal logics, completeness proofs of (hyper)sequent calculi often provide the answer for the decidability problem. Furthermore, the proof search can be quite straightforwardly automatised provided that the calculus is *cut-free*.

Finally, there are proof systems that directly incorporate semantics: in particular, tableaux (e.g., the ones for Gödel logics [2] and tableaux for Łukasiewicz description logic [25]) and labelled sequent calculi (cf., e.g. [32] for labelled sequent calculi for classical modal logics). Because of the calculi's nature, their completeness proofs are usually simple. Besides, the calculi serve as a decision procedure that either establishes that the given formula is valid or provides an explicit countermodel.

Our tableaux system $\mathcal{T}\left(\mathbf{KG}^2_{\mathsf{fb}}\right)$ is a straightforward modal expansion of constraint tableaux for G^2 presented in [5]. It is inspired by constraint tableaux for Łukasiewicz logics from [21,22] (but cf. [26] for an approach similar to ours) which we modify with two-sorted labels corresponding to the support of truth and support of falsity in the model. This idea comes from tableaux for the Belnap—Dunn logic by D'Agostino [1]. Moreover, since $\mathbf{KG}^2_{\mathsf{fb}}$ is a conservative extension of $\mathbf{K}\mathsf{biG}_{\mathsf{fb}}$, our calculus can be used for that logic as well if we apply only the rules that govern the support of truth of $\mathsf{bi}\mathcal{L}_{\Box,\Diamond}$ formulas.

Definition 6 ($\mathcal{T}\left(\mathbf{KG}^2_{\mathsf{fb}}\right)$). *We fix a set of state-labels* W *and let* $\lesssim\in\{<,\leqslant\}$ *and* $\gtrsim\in\{>,\geqslant\}$*. Let further* $w\in\mathsf{W}$, $\mathbf{x}\in\{1,2\}$, $\phi\in\mathsf{bi}\mathcal{L}^{\neg}_{\Box,\Diamond}$, *and* $c\in\{0,1\}$*. A structure is either* $w:\mathbf{x}:\phi$ *or* c*. We denote the set of structures with* Str*.*

We define a constraint tableau *as a downward branching tree whose branches are sets containing the following types of entries:*

- relational constraints *of the form* $w\mathsf{R}w'$ *with* $w,w'\in\mathsf{W}$*;*
- structural constraints *of the form* $\mathfrak{X}\lesssim\mathfrak{X}'$ *with* $\mathfrak{X},\mathfrak{X}'\in\mathsf{Str}$.

Each branch can be extended by an application of a rule[6] *from Fig. 4 or Fig. 5.*

A tableau's branch $\mathcal{B}$ *is* closed *iff one of the following conditions applies:*

- *the transitive closure of* $\mathcal{B}$ *under* $\lesssim$ *contains* $\mathfrak{X}<\mathfrak{X}$*;*
- $0\geqslant1\in\mathcal{B}$, *or* $\mathfrak{X}>1\in\mathcal{B}$, *or* $\mathfrak{X}<0\in\mathcal{B}$.

A tableau is closed *iff all its branches are closed. We say that there is a* tableau proof *of* ϕ *iff there is a closed tableau starting from the constraint* $w:1:\phi<1$*.*

An open branch $\mathcal{B}$ *is* complete *iff the following condition is met.*

$*$ If all premises of a rule occur on $\mathcal{B}$, then its one conclusion[7] occurs on $\mathcal{B}$.

Remark 4. Note that due to Proposition 1, we need to check only one valuation of ϕ to verify its validity.

Convention 4 (Interpretation of constraints). *The following table gives the interpretations of structural constraints on the example of* $\leqslant$*.*

[6] If $\mathfrak{X}<1$ and $\mathfrak{X}<\mathfrak{X}'$ (or $0<\mathfrak{X}'$ and $\mathfrak{X}<\mathfrak{X}'$) occur on $\mathcal{B}$, then the rules are applied only to $\mathfrak{X}<\mathfrak{X}'$.

[7] Note that branching rules have *two* conclusions.

$$\neg_1{\lesssim}\,\frac{w:1:\neg\phi\lesssim\mathfrak{X}}{w:2:\phi\lesssim\mathfrak{X}}\qquad\neg_2{\lesssim}\,\frac{w:2:\neg\phi\lesssim\mathfrak{X}}{w:1:\phi\lesssim\mathfrak{X}}\qquad\neg_1{\gtrsim}\,\frac{w:1:\neg\phi\gtrsim\mathfrak{X}}{w:2:\phi\gtrsim\mathfrak{X}}\qquad\neg_2{\gtrsim}\,\frac{w:2:\neg\phi\gtrsim\mathfrak{X}}{w:1:\phi\gtrsim\mathfrak{X}}$$

$$\wedge_1{\gtrsim}\,\frac{w:1:\phi\wedge\phi'\gtrsim\mathfrak{X}}{\begin{array}{c}w:1:\phi\gtrsim\mathfrak{X}\\ w:1:\phi'\gtrsim\mathfrak{X}\end{array}}\;\wedge_2{\lesssim}\,\frac{w:2:\phi\wedge\phi'\lesssim\mathfrak{X}}{\begin{array}{c}w:2:\phi\lesssim\mathfrak{X}\\ w:2:\phi'\lesssim\mathfrak{X}\end{array}}\;\vee_1{\lesssim}\,\frac{w:1:\phi\vee\phi'\lesssim\mathfrak{X}}{\begin{array}{c}w:1:\phi\lesssim\mathfrak{X}\\ w:1:\phi'\lesssim\mathfrak{X}\end{array}}\;\vee_2{\gtrsim}\,\frac{w:2:\phi\vee\phi'\gtrsim\mathfrak{X}}{\begin{array}{c}w:2:\phi\gtrsim\mathfrak{X}\\ w:2:\phi'\gtrsim\mathfrak{X}\end{array}}$$

$$\wedge_1{\lesssim}\,\frac{w:1:\phi\wedge\phi'\lesssim\mathfrak{X}}{w:1:\phi\lesssim\mathfrak{X}\mid w:1:\phi'\lesssim\mathfrak{X}}\qquad\wedge_2{\gtrsim}\,\frac{w:2:\phi\wedge\phi'\gtrsim\mathfrak{X}}{w:2:\phi\gtrsim\mathfrak{X}\mid w:2:\phi'\gtrsim\mathfrak{X}}$$

$$\vee_1{\gtrsim}\,\frac{w:1:\phi\vee\phi'\gtrsim\mathfrak{X}}{w:1:\phi\gtrsim\mathfrak{X}\mid w:1:\phi'\gtrsim\mathfrak{X}}\qquad\vee_2{\lesssim}\,\frac{w:2:\phi\vee\phi'\lesssim\mathfrak{X}}{w:2:\phi\lesssim\mathfrak{X}\mid w:2:\phi'\lesssim\mathfrak{X}}$$

$$\to_1{\leqslant}\,\frac{w:1:\phi\to\phi'\leqslant\mathfrak{X}}{\begin{array}{c|c}\mathfrak{X}\geqslant1&\begin{array}{c}\mathfrak{X}<1\\ w:1:\phi'\leqslant\mathfrak{X}\\ w:1:\phi>w:1:\phi'\end{array}\end{array}}\qquad\to_1{\gtrsim}\,\frac{w:1:\phi\to\phi'\gtrsim\mathfrak{X}}{w:1:\phi\leqslant w:1:\phi'\mid w:1:\phi'\gtrsim\mathfrak{X}}$$

$$\to_2{\lesssim}\,\frac{w:2:\phi\to\phi'\lesssim\mathfrak{X}}{w:2:\phi'\leqslant w:2:\phi\mid w:2:\phi'\lesssim\mathfrak{X}}\qquad\to_2{\geqslant}\,\frac{w:2:\phi\to\phi'\geqslant\mathfrak{X}}{\begin{array}{c|c}\mathfrak{X}\leqslant0&\begin{array}{c}\mathfrak{X}>0\\ w:2:\phi'\geqslant\mathfrak{X}\\ w:2:\phi'>w:2:\phi\end{array}\end{array}}$$

$$\prec_1{\lesssim}\,\frac{w:1:\phi\prec\phi'\lesssim\mathfrak{X}}{w:1:\phi\leqslant w:1:\phi'\mid w:1:\phi\lesssim\mathfrak{X}}\qquad\prec_1{\geqslant}\,\frac{w:1:\phi\prec\phi'\geqslant\mathfrak{X}}{\begin{array}{c|c}\mathfrak{X}\leqslant0&\begin{array}{c}\mathfrak{X}>0\\ w:1:\phi\geqslant\mathfrak{X}\\ w:1:\phi>w:1:\phi'\end{array}\end{array}}$$

$$\prec_2{\gtrsim}\,\frac{w:2:\phi\prec\phi'\gtrsim\mathfrak{X}}{w:2:\phi\gtrsim\mathfrak{X}\mid w:2:\phi'\leqslant w:2:\phi}\qquad\prec_2{\leqslant}\,\frac{w:2:\phi\prec\phi'\leqslant\mathfrak{X}}{\begin{array}{c|c}\mathfrak{X}\geqslant1&\begin{array}{c}\mathfrak{X}<1\\ w:2:\phi\leqslant\mathfrak{X}\\ w:2:\phi'>w:2:\phi\end{array}\end{array}}$$

$$\to_1{<}\,\frac{w:1:\phi\to\phi'<\mathfrak{X}}{\begin{array}{c}w:1:\phi'<\mathfrak{X}\\ w:1:\phi>w:1:\phi'\end{array}}\;\to_2{>}\,\frac{w:2:\phi\to\phi'>\mathfrak{X}}{\begin{array}{c}w:2:\phi'>\mathfrak{X}\\ w:2:\phi'>w:2:\phi\end{array}}$$

$$\prec_1{>}\,\frac{w:1:\phi\prec\phi'>\mathfrak{X}}{\begin{array}{c}w:1:\phi>\mathfrak{X}\\ w:1:\phi>w:1:\phi'\end{array}}\;\prec_2{<}\,\frac{w:2:\phi\prec\phi'<\mathfrak{X}}{\begin{array}{c}w:2:\phi<\mathfrak{X}\\ w:2:\phi<w:2:\phi'\end{array}}$$

Fig. 4. Propositional rules of $\mathcal{T}\left(\mathbf{KG}^2_{\mathsf{fb}}\right)$. Bars denote branching.

entry	**interpretation**
$w\colon 1\colon \phi \leqslant w'\colon 2\colon \phi'$	$v_1(\phi, w) \leq v_2(\phi', w')$
$w\colon 2\colon \phi \leqslant c$	$v_2(\phi, w) \leq c$ with $c \in \{0, 1\}$

As one can see from Fig. 4 and Fig. 5, the rules follow the semantical conditions from Definition 4. Let us discuss $\rightarrow_1\leqslant$ and $\Box_1\lesssim$ in more details.

The premise of $\rightarrow_1\leqslant$ is interpreted as $v_1(\phi \rightarrow \phi', w) \leqslant x$. To decompose the implication, we check two options: either $x = 1$ (then, the value of $\phi \rightarrow \phi'$ is arbitrary) or $x < 1$. In the second case, we use the semantics to obtain that $v_1(\phi', w) \leqslant x$ and $v_1(\phi, w) > v_1(\phi', w)$.

$$
\Box_1\gtrsim\ \dfrac{\begin{array}{c} w\colon 1\colon \Box\phi \gtrsim \mathfrak{X} \\ wRw' \end{array}}{w'\colon 1\colon \phi \gtrsim \mathfrak{X}}
\quad
\Box_1\lesssim\ \dfrac{w\colon 1\colon \Box\phi \lesssim \mathfrak{X}}{\begin{array}{c} wRw'' \\ w''\colon 1\colon \phi \lesssim \mathfrak{X} \end{array}}
\quad
\Box_2\gtrsim\ \dfrac{w\colon 2\colon \Box\phi \gtrsim \mathfrak{X}}{\begin{array}{c} wRw'' \\ w''\colon 2\colon \phi \gtrsim \mathfrak{X} \end{array}}
\quad
\Box_2\lesssim\ \dfrac{\begin{array}{c} w\colon 2\colon \Box\phi \lesssim \mathfrak{X} \\ wRw' \end{array}}{w'\colon 2\colon \phi \lesssim \mathfrak{X}}
$$

$$
\Diamond_1\lesssim\ \dfrac{\begin{array}{c} w\colon 1\colon \Diamond\phi \lesssim \mathfrak{X} \\ wRw' \end{array}}{w'\colon 1\colon \phi \lesssim \mathfrak{X}}
\quad
\Diamond_1\gtrsim\ \dfrac{w\colon 1\colon \Diamond\phi \gtrsim \mathfrak{X}}{\begin{array}{c} wRw'' \\ w''\colon 1\colon \phi \gtrsim \mathfrak{X} \end{array}}
\quad
\Diamond_2\lesssim\ \dfrac{w\colon 2\colon \Diamond\phi \lesssim \mathfrak{X}}{\begin{array}{c} wRw'' \\ w''\colon 2\colon \phi \lesssim \mathfrak{X} \end{array}}
\quad
\Diamond_2\gtrsim\ \dfrac{\begin{array}{c} w\colon 2\colon \Diamond\phi \gtrsim \mathfrak{X} \\ wRw' \end{array}}{w'\colon 2\colon \phi \gtrsim \mathfrak{X}}
$$

Fig. 5. Modal rules of $\mathcal{T}\left(\mathbf{KG}^2_{\mathsf{fb}}\right)$. w'' is fresh on the branch.

In order to apply $\Box_1\lesssim$ to $w\colon 1\colon \Box\phi \lesssim \mathfrak{X}$, we introduce a new state w'' that is seen by w. Since we work in a finite branching model, w'' can witness the value of $\Box\phi$. Thus, we add $w''\colon 1\colon \phi \lesssim \mathfrak{X}$.

We also provide an example of how our tableaux work. On Fig. 6, one can see a successful proof on the left and a failed proof on the right.

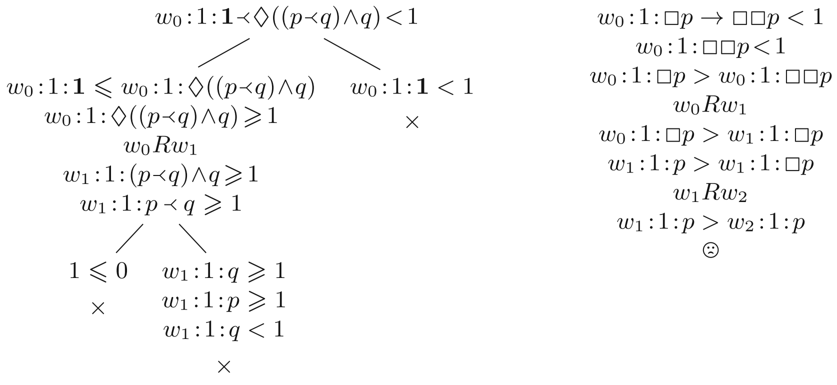

$w_0\colon 1\colon \mathbf{1} \prec \Diamond((p \prec q) \wedge q) < 1$
$w_0\colon 1\colon \mathbf{1} \leqslant w_0\colon 1\colon \Diamond((p \prec q) \wedge q)$
$w_0\colon 1\colon \Diamond((p \prec q) \wedge q) \geqslant 1$
$w_0 R w_1$
$w_1\colon 1\colon (p \prec q) \wedge q \geqslant 1$
$w_1\colon 1\colon p \prec q \geqslant 1$
$1 \leqslant 0$
×
$w_1\colon 1\colon q \geqslant 1$
$w_1\colon 1\colon p \geqslant 1$
$w_1\colon 1\colon q < 1$
×
$w_0\colon 1\colon \mathbf{1} < 1$
×
$w_0\colon 1\colon \Box p \rightarrow \Box\Box p < 1$
$w_0\colon 1\colon \Box\Box p < 1$
$w_0\colon 1\colon \Box p > w_0\colon 1\colon \Box\Box p$
$w_0 R w_1$
$w_0\colon 1\colon \Box p > w_1\colon 1\colon \Box p$
$w_1\colon 1\colon p > w_1\colon 1\colon \Box p$
$w_1 R w_2$
$w_1\colon 1\colon p > w_2\colon 1\colon p$
☹

Fig. 6. × indicates closed branches; ☹ indicates complete open branches.

Definition 7 (Branch realisation). *We say that a model $\mathfrak{M} = \langle W, R, v_1, v_2\rangle$ with $W = \{w : w$ occurs on $\mathcal{B}\}$ and $R = \{\langle w, w'\rangle : w\mathsf{R}w' \in \mathcal{B}\}$* realises a branch *$\mathcal{B}$ of a tree iff the following conditions are met.*

- *$v_{\mathbf{x}}(\phi, w) \leq v_{\mathbf{x}'}(\phi', w')$ for any $w : \mathbf{x} : \phi \leqslant w' : \mathbf{x}' : \phi' \in \mathcal{B}$ with $\mathbf{x}, \mathbf{x}' \in \{1, 2\}$.*
- *$v_{\mathbf{x}}(\phi, w) \leq c$ for any $w : \mathbf{x} : \phi \leqslant c \in \mathcal{B}$ with $c \in \{0, 1\}$.*

Theorem 3 (Completeness). *ϕ is $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$ valid iff it has a $\mathcal{T}(\mathbf{K}\mathsf{G}^2_{\mathsf{fb}})$ proof.*

Proof. We consider only the $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$ case since $\mathbf{K}\mathsf{biG}_{\mathsf{fb}}$ can be handled the same way. For soundness, we check that if the premise of the rule is realised, then so is at least one of its conclusions. We consider the cases of $\rightarrow_1\leqslant$ and $\Box_1\lesssim$. Assume that $w : 1 : \phi \rightarrow \phi' \leqslant \mathfrak{X}$ is realised and assume w.l.o.g. that $\mathfrak{X} = u : 2 : \psi$. It is clear that either $v_2(\psi, u) = 1$ or $v_2(\psi, u) < 1$. In the first case, $\mathfrak{X} \geqslant 1$ is realised. In the second case, we have that $v_1(\phi, w) > v_1(\phi', w)$ and $v_1(\phi', w) \leqslant v_2(\psi, u)$. Thus, $\mathfrak{X} < 1$, $w : 1 : \phi > w : 1 : \phi'$, and $w : 1 : \phi' \leqslant u : 1 : \psi$ are realised as well, as required.

For $\Box_1\lesssim$, assume that $w : 1 : \Box\phi \leqslant \mathfrak{X}$ is realised and assume w.l.o.g. that $\mathfrak{X} = u : 2 : \psi$. Thus, $v_1(\Box\phi, w) \leqslant v_2(\psi, u)$ Then, since the model is finitely branching, there is an accessible state w'' s.t. $v_1(\phi, w) \leqslant v_2(\psi, u)$. Thus, $w'' : 1 : \phi \leqslant \mathfrak{X}$ is realised too.

As no closed branch is realisable, the result follows.

For completeness, we show that every complete open branch $\mathcal{B}$ is realisable. We construct the model as follows. We let $W = \{w : w$ occurs in $\mathcal{B}\}$, and set $R = \{\langle w, w'\rangle : w\mathsf{R}w' \in \mathcal{B}\}$. Now, it remains to construct the suitable valuations.

For $i \in \{1, 2\}$, if $w : i : p \geqslant 1 \in \mathcal{B}$, we set $v_i(p, w) = 1$. If $w : i : p \leqslant 0 \in \mathcal{B}$, we set $v_i(p, w) = 0$. To set the values of the remaining variables $q_1, \ldots, q_n$, we proceed as follows. Denote $\mathcal{B}^+$ the transitive closure of $\mathcal{B}$ under $\lesssim$ and let

$$[w\!:\!\mathbf{x}\!:\!q_i] = \left\{ w'\!:\!\mathbf{x}'\!:\!q_j \;\middle|\; \begin{array}{c} w\!:\!\mathbf{x}\!:\!q_i \leqslant w'\!:\!\mathbf{x}'\!:\!q_j \in \mathcal{B}^+ \text{ and } w\!:\!\mathbf{x}\!:\!q_i < w'\!:\!\mathbf{x}'\!:\!q_j \notin \mathcal{B}^+ \\ \text{or} \\ w\!:\!\mathbf{x}\!:\!q_i \geqslant w'\!:\!\mathbf{x}'\!:\!q_j \in \mathcal{B}^+ \text{ and } w\!:\!\mathbf{x}\!:\!q_i > w'\!:\!\mathbf{x}'\!:\!q_j \notin \mathcal{B}^+ \end{array} \right\}$$

It is clear that there are at most $2 \cdot n \cdot |W|$ $[w : \mathbf{x} : q_i]$'s since the only possible loop in $\mathcal{B}^+$ is $w_{i_1}\!:\!\mathbf{x}\!:\!r \leqslant \ldots \leqslant w_{i_1}\!:\!\mathbf{x}\!:\!r$, but in such a loop all elements belong to $[w_{i_1}\!:\!\mathbf{x}\!:\!r]$. We put $[w\!:\!\mathbf{x}\!:\!q_i] \prec [w'\!:\!\mathbf{x}'\!:\!q_j]$ iff there are $w_k\!:\!\mathbf{x}\!:\!r \in [w\!:\!\mathbf{x}\!:\!q_i]$ and $w'_k\!:\!\mathbf{x}'\!:\!r' \in [w'\!:\!\mathbf{x}'\!:\!q_j]$ s.t. $w_k\!:\!\mathbf{x}\!:\!r < w'_k\!:\!\mathbf{x}'\!:\!r' \in \mathcal{B}^+$.

We now set the valuation of these variables as follows

$$v_{\mathbf{x}}(q_i, w) = \frac{|\{[w'\!:\!\mathbf{x}'\!:\!q'] \mid [w'\!:\!\mathbf{x}'\!:\!q'] \prec [w\!:\!\mathbf{x}\!:\!q_i]\}|}{2 \cdot n \cdot |W|}$$

Note that if some ϕ contains s but $\mathcal{B}^+$ contains no inequality with it, the above definition ensures that s is going to be evaluated at 0. Thus, all constraints containing only variables are satisfied.

It remains to show that all other constraints are satisfied. For that, we prove that if at least one conclusion of the rule is satisfied, then so is the premise. The propositional cases are straightforward and can be tackled in the same manner as in [5, Theorem 2]. We consider only the case of $\lozenge_2 \gtrsim$. Assume w.l.o.g. that $\gtrsim = \geqslant$ and $\mathfrak{X} = u:1:\psi$. Since $\mathcal{B}$ is complete, if $w: 2:\lozenge\phi \geqslant u:1:\psi \in \mathcal{B}$, then for any w' s.t. $w\mathsf{R}w' \in \mathcal{B}$, we have $w': 2:\phi \geqslant u:1:\psi \in \mathcal{B}$, and all of them are realised by $\mathfrak{M}$. But then $w: 2:\lozenge\phi \geqslant u:1:\psi$ is realised too, as required.

Theorem 4.

1. *Let* $\phi \in \mathsf{bi}\mathcal{L}^{\neg}_{\Box,\lozenge}$ *be* not $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$ valid, *and let* $|\phi|$ *denote the number of symbols in it. Then there is a model* $\mathfrak{M}$ *of the size* $O(|\phi|^{|\phi|})$ *and depth* $O(|\phi|)$ *and* $w \in \mathfrak{M}$ *s.t.* $v_1(\phi, w) \neq 1$.
2. $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$ *validity and satisfiability*[8] *are* PSPACE*-complete.*

Proof. We begin with 1. By Theorem 3, if ϕ is *not* $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$ *valid*, we can build a falsifying model using tableaux. It is also clear from the rules on Fig. 5 that the depth of the constructed model is bounded from above by the maximal number of nested modalities in ϕ. The width of the model is bounded by the maximal number of modalities on the same level of nesting. The sharpness of the bound is obtained using the embedding of $\mathbf{K}$ into $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$ since $\mathbf{K}$ is complete w.r.t. finitely branching models and it is possible to force shallow trees of exponential size in $\mathbf{K}$ (cf., e.g. [7, §6.7]). The embedding also entails PSPACE-hardness. It remains to tackle membership.

First, observe from the proof of Theorem 3 that $\phi(p_1, \ldots, p_n)$ is satisfiable (falsifiable) on $\mathfrak{M} = \langle W, R, v_1, v_2\rangle$ iff there are v_1 and v_2 that give variables values from $\mathsf{V} = \left\{0, \frac{1}{2\cdot n\cdot|W|}, \ldots, \frac{2\cdot n\cdot|W|-1}{2\cdot n\cdot|W|}, 1\right\}$ under which ϕ is satisfied (falsified).

As we mentioned, $|W|$ is bounded from above by k^{k+1} with k being the number of modalities in ϕ. Therefore, we replace structural constraints with labelled formulas of the form $w:i:\phi = \mathsf{v}$ ($\mathsf{v} \in \mathsf{V}$) avoiding comparisons of values of formulas in different states. As expected, we close the branch if it contains $w:i:\psi = \mathsf{v}$ and $w:i:\psi = \mathsf{v}'$ for $\mathsf{v} \neq \mathsf{v}'$.

Now we replace the rules with the new ones that work with labelled formulas instead of structural constraints. Below, we give as an example new rules for → and $\lozenge$[9] (with $|\mathsf{V}| = m+1$):

$$\dfrac{w:1:\phi \to \phi' = 1}{w:1:\phi = 0 \left| \begin{array}{l} w:1:\phi = \frac{1}{m+1} \\ w:1:\phi' = \frac{1}{m+1} \end{array} \right| \begin{array}{l} w:1:\phi = \frac{1}{m+1} \\ w:1:\phi' = \frac{2}{m+1} \end{array} \Bigg| \ldots \Bigg| \begin{array}{l} w:1:\phi = \frac{m-1}{m+1} \\ w:1:\phi' = \frac{m}{m+1} \end{array} \Bigg| \; w:1:\phi' = 1}$$

$$\dfrac{w:1:\lozenge\phi = \frac{r}{m+1}}{w\mathsf{R}w''; w'':1:\phi = \frac{r}{m+1}} \qquad \dfrac{w:1:\lozenge\phi = \frac{r}{m+1}; w\mathsf{R}w'}{w':1:\phi = 0 \mid \ldots \mid w':1:\phi = \frac{r-1}{m+1}}$$

[8] Satisfiability and falsifiability (non-validity) are reducible to each other using $\prec$: ϕ is satisfiable iff $\sim\sim(\phi \prec \mathbf{0})$ is falsifiable; ϕ is falsifiable iff $\sim\sim(\mathbf{1} \prec \phi)$ is satisfiable.

[9] Intuitively, for a value $1 > \mathsf{v} > 0$ of $\lozenge\phi$ at w, we add a new state that witnesses v, and for a state on the branch, we guess a value smaller than v. Other modal rules can be rewritten similarly.

We now show how to build a satisfying model for ϕ using polynomial space. We begin with $w_0:1:\phi=1$ and start applying propositional rules (first, those that do not require branching). If we implement a branching rule, we pick one branch and work only with it: either until the branch is closed, in which case we pick another one; until no more rules are applicable (then, the model is constructed); or until we need to apply a modal rule to proceed. At this stage, we need to store only the subformulas of ϕ with labels denoting their value at w_0.

Now we guess a modal formula (say, $w_0:2:\Box\chi=\frac{1}{m+1}$) whose decomposition requires an introduction of a new state (w_1) and apply this rule. Then we apply all modal rules that use $w_0\mathsf{R}w_1$ as a premise (again, if those require branching, we guess only one branch) and start from the beginning with the propositional rules. If we reach a contradiction, the branch is closed. Again, the only new entries to store are subformulas of ϕ (now, with fewer modalities), their values at w_1, and a relational term $w_0\mathsf{R}w_1$. Since the depth of the model is $O(|\phi|)$ and since we work with modal formulas one by one, we need to store subformulas of ϕ with their values $O(|\phi|)$ times, so, we need only $O(|\phi|^2)$ space.

Finally, if no rule is applicable and there is no contradiction, we mark $w_0:2:\Box\chi=\frac{1}{m+1}$ as 'safe'. Now we *delete all entries of the tableau below it* and pick another unmarked modal formula that requires an introduction of a new state. Dealing with these one by one allows us to construct the model branch by branch. But since the length of each branch of the model is bounded by $O(|\phi|)$ and since we delete *branches of the model* once they are shown to contain no contradictions, we need only polynomial space.

We end the section with two simple observations. First, Theorems 3 and 4 are applicable both to $\mathbf{K}\mathsf{biG}_{\mathsf{fb}}$ and $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$ because the latter is conservative over the former. Secondly, since $\mathbf{K}\mathsf{G}^2$ and $\mathbf{K}\mathsf{biG}$ are conservative over $\mathfrak{GK}^c$ and since $\mathbf{K}$ can be embedded in $\mathfrak{GK}^c$, the lower bounds on complexity of a classical modal logic of some class of frames $\mathbb{K}$ and G^2 modal logic of $\mathbb{K}$ will coincide.

5 Concluding Remarks

In this paper, we developed a crisp modal expansion of the two-dimensional Gödel logic G^2 as well as an expansion of bi-Gödel logic with $\Box$ and $\Diamond$ both for crisp and fuzzy frames. We also established their connections with modal Gödel logics, and gave a complexity analysis of their finitely branching fragments.

The following steps are: to study the proof theory of $\mathbf{K}\mathsf{G}^2$ and $\mathbf{K}\mathsf{G}^2_{\mathsf{fb}}$: both in the form of Hilbert-style and sequent calculi; establish the decidability (or lack thereof) for the case of $\mathbf{K}\mathsf{G}^2$. Moreover, two-dimensional treatment of information invites for different modalities, e.g. those formalising aggregation strategies given in [6]—in particular, the cautious one (where the agent takes minima/infima of *both* positive and negative supports of a given statement) and the confident one (whereby the maxima/suprema are taken). Last but not least, while in this paper we assumed that our *access* to sources is crisp, one can argue that the *degree of our bias* towards the given source can be formalised via *fuzzy* frames. Thus, it would be instructive to construct a fuzzy version of $\mathbf{K}\mathsf{G}^2$.

In a broader perspective, we plan to provide a general treatment of two-dimensional modal logics of uncertainty. Indeed, within our project [5,6], we are formalising reasoning with heterogeneous and possibly incomplete and inconsistent information (such as crisp or fuzzy data, personal beliefs, etc.) in a modular fashion. This modularity is required because different contexts should be treated with different logics—indeed, not only the information itself can be of various nature but the reasoning strategies of different agents even applied to the same data are not necessarily the same either. Thus, since we wish to account for this diversity, we should be able to combine different logics in our approach.

References

1. Agostino, M.D.: Investigations into the complexity of some propositional calculi. Oxford University Computing Laboratory, Oxford (1990)
2. Avron, A., Konikowska, B.: Decomposition proof systems for Gödel-Dummett logics. Stud. Logica **69**(2), 197–219 (2001)
3. Baaz, M.: Infinite-valued Gödel logics with 0-1-projections and relativizations. In: Gödel 1996: Logical Foundations of Mathematics, Computer Science and Physics–Kurt Gödel's legacy, Brno, Czech Republic, August 1996, Proceedings, pp. 23–33. Association for Symbolic Logic (1996)
4. Belnap, Nuel D..: How a computer should think. In: Omori, H., Wansing, H. (eds.) New Essays on Belnap-Dunn Logic. SL, vol. 418, pp. 35–53. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31136-0_4
5. Bílková, M., Frittella, S., Kozhemiachenko, D.: Constraint tableaux for two-dimensional fuzzy logics. In: Das, A., Negri, S. (eds.) TABLEAUX 2021. LNCS (LNAI), vol. 12842, pp. 20–37. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86059-2_2
6. Bílková, M., Frittella, S., Majer, O., Nazari, S.: Belief based on inconsistent information. In: Martins, M.A., Sedlár, I. (eds.) Dynamic Logic. New Trends and Applications, pp. 68–86. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65840-3_5
7. Blackburn, P., Rijke, M.D., Venema, Y.: Modal logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, 4. print. with corr. edn. (2010)
8. Bobillo, F., Delgado, M., Gómez-Romero, J., Straccia, U.: Fuzzy description logics under Gödel semantics. Int. J. Approx. Reason. **50**(3), 494–514 (2009)
9. Bobillo, F., Delgado, M., Gómez-Romero, J., Straccia, U.: Joining Gödel and Zadeh fuzzy logics in fuzzy description logics. Int. J. Uncertain. Fuzziness Knowledge-Based Syst. **20**(04), 475–508 (2012)
10. Borgwardt, S., Distel, F., Peñaloza, R.: Decidable Gödel description logics without the finitely-valued model property. In: Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning (2014)
11. Caicedo, X., Metcalfe, G., Rodríguez, R., Rogger, J.: A finite model property for Gödel modal logics. In: Libkin, L., Kohlenbach, U., de Queiroz, R. (eds.) WoLLIC 2013. LNCS, vol. 8071, pp. 226–237. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39992-3_20
12. Caicedo, X., Rodriguez, R.: Standard Gödel modal logics. Stud. Logica **94**(2), 189–214 (2010). https://doi.org/10.1007/s11225-010-9230-1

13. Caicedo, X., Rodríguez, R.: Bi-modal Gödel logic over [0,1]-valued Kripke frames. J. Log. Comput. **25**(1), 37–55 (2015)
14. Chagrov, A., Zakharyaschev, M.: Modal Logic. Clarendon Press, Oxford (1997)
15. Cintula, P., Noguera, C.: Modal logics of uncertainty with two-layer syntax: a general completeness theorem. In: Kohlenbach, U., Barceló, P., de Queiroz, R. (eds.) WoLLIC 2014. LNCS, vol. 8652, pp. 124–136. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44145-9_9
16. Dunn, J.M.: Intuitive semantics for first-degree entailments and 'coupled trees'. Philos. Stud. **29**(3), 149–168 (1976)
17. Ertola, R., Esteva, F., Flaminio, T., Godo, L., Noguera, C.: Paraconsistency properties in degree-preserving fuzzy logics. Soft Comput. **19**, 1–16 (2014). https://doi.org/10.1007/s00500-014-1489-0
18. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge (2003)
19. Ginsberg, M.: Multivalued logics: a uniform approach to reasoning in AI. Comput. Intell **4**, 256–316 (1988)
20. Grigolia, R., Kiseliova, T., Odisharia, V.: Free and projective bimodal symmetric gödel algebras. Stud. Logica **104**(1), 115–143 (2016)
21. Hähnle, R.: A new translation from deduction into integer programming. In: Calmet, J., Campbell, J.A. (eds.) AISMC 1992. LNCS, vol. 737, pp. 262–275. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57322-4_18
22. Hähnle, R.: Many-valued logic and mixed integer programming. Ann. Math. Artif. Intell. **12**(3–4), 231–263 (1994)
23. Hájek, P.: Metamathematics of Fuzzy Logic. Trends in Logic, 4th edn. Springer, Dordrecht (1998). https://doi.org/10.1007/978-94-011-5300-3
24. Jansana, R., Rivieccio, U.: Residuated bilattices. Soft. Comput. **16**(3), 493–504 (2012)
25. Kułacka, A., Pattinson, D., Schröder, L.: Syntactic labelled tableaux for Łukasiewicz fuzzy ALC. In: Twenty-Third International Joint Conference on Artificial Intelligence. AAAI Press (2013)
26. Lascio, L.D., Gisolfi, A.: Graded tableaux for rational Pavelka logic. Int. J. Intell. Syst. **20**(12), 1273–1285 (2005)
27. Leitgeb, H.: Hype: a system of hyperintensional logic (with an application to semantic paradoxes). J. Philos. Log. **48**(2), 305–405 (2019)
28. Metcalfe, G., Olivetti, N.: Proof systems for a Gödel modal logic. In: Giese, M., Waaler, A. (eds.) TABLEAUX 2009. LNCS (LNAI), vol. 5607, pp. 265–279. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02716-1_20
29. Metcalfe, G., Olivetti, N.: Towards a proof theory of Gödel modal logics. Log. Methods Comput. Sci. **7** (2011)
30. Metcalfe, G., Olivetti, N., Gabbay, D.: Proof Theory for Fuzzy Logics. Applied Logic Series, vol. 36. Springer, Dordrecht (2008). https://doi.org/10.1007/978-1-4020-9409-5
31. Moisil, G.: Logique modale. Disquisitiones mathematicae et physicae **2**, 3–98 (1942)
32. Negri, S.: Proof analysis in modal logic. J. Philos. Log. **34**(5–6), 507–544 (2005)
33. Odintsov, S., Wansing, H.: Routley star and hyperintensionality. J. Philos. Log. **50**, 33–56 (2021)
34. Omori, H., Wansing, H.: 40 years of FDE: an introductory overview. Stud. Logica. **105**(6), 1021–1049 (2017). https://doi.org/10.1007/s11225-017-9748-6
35. Rivieccio, U.: An algebraic study of bilattice-based logics. Ph.D. thesis, University of Barcelona – University of Genoa (2010)

36. Rodriguez, R.O., Vidal, A.: Axiomatization of Crisp Gödel Modal Logic. Stud. Logica **109**(2), 367–395 (2020). https://doi.org/10.1007/s11225-020-09910-5
37. Vakarelov, D.: Notes on N-lattices and constructive logic with strong negation. Stud. Logica **36**(1–2), 109–125 (1977)
38. Wansing, H.: Constructive negation, implication, and co-implication. J. Appl. Non-Classical Logics **18**(2–3), 341–364 (2008). https://doi.org/10.3166/jancl.18.341-364

Non-associative, Non-commutative Multi-modal Linear Logic

Eben Blaisdell[1], Max Kanovich[2], Stepan L. Kuznetsov[3,4], Elaine Pimentel[2(✉)], and Andre Scedrov[1]

[1] Department of Mathematics, University of Pennsylvania, Philadelphia, USA
[2] Department of Computer Science, University College London, London, UK
elaine.pimentel@gmail.com
[3] Steklov Mathematical Institute of RAS, Moscow, Russia
[4] Faculty of Computer Science, HSE University, Moscow, Russia

Abstract. Adding multi-modalities (called *subexponentials*) to linear logic enhances its power as a logical framework, which has been extensively used in the specification of *e.g.* proof systems, programming languages and bigraphs. Initially, subexponentials allowed for classical, linear, affine or relevant behaviors. Recently, this framework was enhanced so to allow for commutativity as well. In this work, we close the cycle by considering associativity. We show that the resulting system (acLL_Σ) admits the (multi)cut rule, and we prove two undecidability results for fragments/variations of acLL_Σ.

1 Introduction

Resource aware logics have been object of passionate study for quite some time now. The motivations for this passion vary: resource consciousness are adequate for modeling steps of computation; logics have interesting algebraic semantics; calculi have nice proof theoretic properties; multi-modalities allow for the specification of several behaviors; there are many interesting applications in linguistics, etc.

With this variety of subjects, applications and views, it is not surprising that different groups developed different systems based on different principles. For example, the Lambek calculus (L) [29] was introduced for mathematical modeling of natural language syntax, and it extends a basic categorial grammar [3,4] by a concatenation operator. Linear logic (LL) [16], originally discovered by Girard from a semantical analysis of the models of polymorphic λ-calculus, turned out to be a refinement of classical and intuitionistic logic, having the dualities of the former and constructive properties of the

The work of Max Kanovich was partially supported by EPSRC Programme Grant EP/R006865/1: “Interface Reasoning for Interacting Systems (IRIS).”
The work of Stepan L. Kuznetsov was supported by the Theoretical Physics and Mathematics Advancement Foundation “BASIS” and partially performed within the framework of HSE University Basic Research Program and within the project MK-1184.2021.1.1 “Algorithmic and Semantic Questions of Modal Extensions of Linear Logic” funded by the Ministry of Science and Higher Education of Russia.
Elaine Pimentel acknowledges partial support by the MOSAIC project (EU H2020-MSCA-RISE-2020 Project 101007627).

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 449–467, 2022.
https://doi.org/10.1007/978-3-031-10769-6_27

latter. The key point is the presence of the *modalities* $!, ?$, called *exponentials* in LL. In the intuitionistic version of LL, denoted by ILL, only the $!$ exponential is present.

L and LL were compared in [2], when Abrusci showed that Lambek calculus coincides with a variant of the non-commutative, multiplicative version of ILL [41]. This correspondence can be lifted for considering also the additive connectives: Full (multiplicative-additive) Lambek calculus FL relates to non-commutative multiplicative-additive version of ILL, here denoted by cLL.

In this paper we propose the sequent based system acLL_Σ, a conservative extension of cLL, where associativity is allowed only for formulas marked with a special kind of modality, determined by a *subexponential signature* Σ. The notation adopted is modular, uniform and scalable, in the sense that many well known systems will appear as fragments or special cases of acLL_Σ, by only modifying the signature Σ. The core fragment of acLL_Σ (*i.e.*, without the subexponentials) corresponds to the non-associative version of full Lambek calculus, FNL [8].[1]

The language of acLL_Σ consists of a denumerable infinite set of propositional variables $\{p, q, r, \ldots\}$, the unities $\{1, \top\}$, the binary connectives for additive conjunction and disjunction $\{\&, \oplus\}$, the non-commutative multiplicative conjunction $\otimes$, the non-commutative linear implications $\{\rightarrow, \leftarrow\}$, and the unary subexponentials $!^i$, with i belonging to a pre-ordered set of labels $(I, \preceq)$.

Roughly speaking, subexponentials [13] are substructural multi-modalities. In LL, $!\,A$ indicates that the linear formula A behaves *classically*, that is, it can be contracted *and* weakened. Labeling $!$ with indices allows moving one step further: The set I can be partitioned so that, in $!^i A$, A can be contracted *and/or* weakened. This allows for two other types of behavior (other than classical or linear): affine (only weakening) or relevant (only contraction). Pre-ordering the labels (together with an upward closeness requirement) guarantees cut-elimination [42]. But then, why consider only weakening and contraction? Why not also take into account other structural properties, like commutativity or associativity? In [20,21] commutativity was added to the picture, so that in $!^i A$, A can be contracted, weakened, classical or linear, but it may also commute with the neighbor formula. In this work we consider the last missing part: Associativity.

Smoothly extending cLL to allow consideration of the non-associative case is non trivial. This requires a structural recasting/reframing of sequents: we pass from sets/multisets to lists in the non-commutative case, onto trees in the case of non-associativity [28]. As a consequence, the inference rules should act deeply over formulas in tree-structured sequents, which can be tricky in the presence of modalities [17].

On the other side, the multi-modal Lambek calculus introduced in [35,45] and extended/compiled/implemented in [18,36–38][2] use different *families of connectives and contexts*, distinguished by means of indices, or *modes*. Contexts are indexed binary trees, with formulas built from the indexed adjoint connectives $\{\rightarrow_i, \leftarrow_i\}$ and $\otimes_i$ (*e.g.*

[1] The multiplicative fragment of acLL_Σ is the non-associative version of Lambek's calculus, NL, introduced by Lambek himself in [30]. Both the associative calculus L and the non-associative calculus NL have their advantages and disadvantages for the analysis of natural language syntax, as we discuss in more detail in Sect. 2.2.

[2] The Grail family of theorem provers [37] works with a variety of modern type-logical frameworks, including multimodal type-logical grammars.

$(A \to_i B, (C \otimes_j D, H)^k)^i)$. Each mode has its own set of logical rules (following the same rule scheme), and different structural features can be combined via the mode information on the formulas. This gives to the resulting system a multi-modal flavor, but it also results in a language of *binary connectives*, determined by the modes. This forces an unfortunate second level synchronization between implications and tensor, and modalities act over whole *sequents*, not on single *formulas*.

In order to attribute particular resource management properties to individual resources, in [27,33] explicit (classical) multi-modalities $\Diamond_i, \Box_i$ were proposed. While such unary modalities were inspired in LL exponentials, the resemblance stops there. First of all, the logical connectives come together with structural constructors for contexts, which turns $\Diamond_i, \Box_i$ into truncated forms of product and implication.

Second, $\Diamond_i, \Box_i$ have a *temporal behavior*, in the sense that $\Diamond\Box F \Rightarrow F$ and $F \Rightarrow \Box\Diamond F$, which are not provable in LL using the "natural interpretation" $\Diamond = ?$, $\Box = !$.

In this paper, multi-modality is *totally local*, given by the subexponentials. The signature Σ contains the pre-ordered set of labels, together with a function stating which axioms, among weakening, contraction, exchange and associativity, are assumed for each label. Sequents will have a *nested structure*, corresponding to trees of formulas. And rules will be applied deeply in such structures. This not only gives the LL based system a more modern presentation (based on nested systems, like *e.g.* in [10,15]), but it also brings the notation closer to the one adopted by the Lambek community, like in [25]. Finally, it also uniformly extends several LL based systems present in the literature, as Example 8 in the next section shows.

Designing a good system serves more than simple pure proof-theoretic interests: Well behaved, neat proof systems can be used in order to approach several important problems, such as interpolation, complexity and decidability. And decidability of extensions/variants/fragments of L and LL is a fascinating subject of study, since the presence or absence of substructural properties/connectives may completely change the outcome. Indeed, it is well known that LL is undecidable [32], but adding weakening (affine LL) turns the system decidable [24], while removing the additives (MELL – multiplicative, exponential LL) reaches the border of knowledge: It is a long standing open problem [50]. Non-associativity also alters decidability and complexity: L is NP-complete [47], while NL is decidable in polynomial time [1,6]. Finally, the number of subexponentials also plays a role in decision problems: MELL with two subexponentials is undecidable [9].

In this work, we will present two undecidability results, all orbiting (but not encompassing) MELL/FNL. First, we show that acLL_Σ containing the multiplicatives $\otimes, \to$, the additive $\oplus$ and one classical subexponential (allowing contraction and weakening) is undecidable. This is a refinement of the unpublished result by Tanaka [51], which states that FNL plus one fully-powered subexponential is undecidable.

In the second undecidability result, we keep two subexponentials, but with a minimalist configuration: the implicational fragment of the logic plus two subexponentials: the "main" one allowing for contraction, exchange, and associativity (weakening is optional), and an "auxiliary" one allowing only associativity. This is a variation of Chaudhuri's result (in the non-associative, non-commutative case), making use of fewer connectives (tensor is not needed) and less powerful subexponentials.

Table 1. Acronyms/decidability of systems mentioned in the paper.

Acronym	System	Decidable?
L	Lambek calculus	✓
LL	(propositional) linear logic	✗
ILL	intuitionistic LL	✗
MALL	multiplicative-additive LL	✓
iMALL	intuitionistic MALL	✓
FL	full (multiplicative-additive) L	✓
cLL	non-commutative iMALL	✓
acLL_Σ	non-commutative, non-associative ILL with subexponentials	–
NL	non-associative L	✓
FNL	full (multiplicative-additive) NL	✓
MELL	multiplicative-exponential LL	unknown
SDML	simply dependent multimodal linear logics	–
SMALC_Σ	FL with subexponentials	–

The rest of the paper is organized as follows: Sect. 2 presents the system acLL_Σ, showing that it has the cut-elimination property and presenting an example in linguistics; Sect. 3 shows the undecidability results; and Sect. 4 concludes the paper.

We have placed, in Table 1, the acronyms for and decidability of all considered systems. Decidability for the cases marked with "–" depends on the signature Σ.

2 A Nested System for Non-associativity

Similar to modal connectives, the exponential ! in ILL is not *canonical* [13], in the sense that if $i \neq j$ then $!^i F \not\equiv !^j F$. Intuitively, this means that we can mark the exponential with *labels* taken from a set I organized in a pre-order $\preceq$ (*i.e.*, reflexive and transitive), obtaining (possibly infinitely-many) exponentials ($!^i$ for $i \in I$). Also as in multi-modal systems, the pre-order determines the provability relation: for a general formula F, $!^b F$ *implies* $!^a F$ iff $a \preceq b$.

The algebraic structure of subexponentials, combined with their intrinsic structural property allow for the proposal of rich linear logic based frameworks. This opened a venue for proposing different multi-modal substructural logical systems, that encountered a number of different applications. Originally [42], subexponentials could assume only weakening and contraction axioms:

$$\mathsf{C}: \quad !^i F \to !^i F \otimes !^i F \qquad \mathsf{W}: \quad !^i F \to 1$$

This allows the specification of systems with multiple contexts, which may be represented by sets or multisets of formulas [44], as well as the specification and verification of concurrent systems [43], and biological systems [46]. In [20,21], non-commutative systems allowing commutative subexponentials were presented:

$$\mathsf{E}: \quad (!^i F) \otimes G \equiv G \otimes (!^i F)$$

and this has many applications, *e.g.*, in linguistics [21].

In this work, we will present a non-commutative, non-associative linear logic based system, and add the possibility of assuming associativity[3]

$$\mathsf{A1}: \ !^i F \otimes (G \otimes H) \equiv (!^i F \otimes G) \otimes H \qquad \mathsf{A2}: \ (G \otimes H) \otimes !^i F \equiv G \otimes (H \otimes !^i F)$$

as well as commutativity and other structural properties.

We start by presenting an adaption of simply dependent multimodal linear logics (SDML) appearing in [31] to the non-associative/commutative case.

The language of non-commutative SDML is that of (propositional intuitionistic) linear logic with subexponentials [21] supplied with the *left residual*; or similarly, that of FL with subexponentials. Non-associative contexts will be organized via binary trees, here called *structures*.

Definition 1 (Structured sequents). Structures *are formulas or pairs containing structures:*

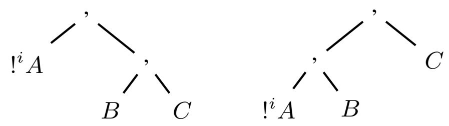
$$\Gamma, \Delta := F \mid (\Gamma, \Gamma)$$

where the constructors may be empty but never a singleton.

An n-ary context $\Gamma\left\{{}^{1}\right\}\ldots\left\{{}^{n}\right\}$ *is a context that contains* n *pairwise distinct numbered* holes $\{\ \}$ *wherever a formula may otherwise occur. Given* n *contexts* $\Gamma_1, \ldots, \Gamma_n$*, we write* $\Gamma\{\Gamma_1\} \cdots \{\Gamma_n\}$ *for the context where the k-th hole in* $\Gamma\left\{{}^{1}\right\}\ldots\left\{{}^{n}\right\}$ *has been replaced by* Γ_k *(for* $1 \leq k \leq n$*). If* $\Gamma_k = \varnothing$ *the hole is removed.*

A structured sequent *(or simply* sequent*) has the form* $\Gamma \Rightarrow F$ *where* Γ *is a structure and* F *is a formula.*

Example 2. Structures are binary trees, with formulas as leaves and commas as nodes. The structure $!^i A, (B, C)$ represents the tree below left, while $(!^i A, B), C$ represents the tree below right



Definition 3 (SDML). *Let* $\mathcal{A}$ *be a set of axioms. A (non-associative/commutative)* simply dependent multimodal logical system *(*SDML*) is given by a triple* $\Sigma = (I, \preccurlyeq, f)$*, where* I *is a set of indices,* $(I, \preccurlyeq)$ *is a pre-order, and* f *is a mapping from* I *to* $2^{\mathcal{A}}$*.*

If Σ *is a* SDML*, then the* logic described by Σ *has the modality* $!^i$ *for every* $i \in I$*, with the rules of* FNL *depicted in Fig. 1, together with rules for the axioms* $f(i)$ *and the interaction axioms* $!^j A \rightarrow !^i A$ *for every* $i, j \in I$ *with* $i \preccurlyeq j$*. Finally, every* SDML *is assumed to be upwardly closed w.r.t.* $\preceq$*, that is, if* $i \preceq j$ *then* $f(i) \subseteq f(j)$ *for all* $i, j \in I$*.*

[3] Note that the implemented rules in Fig. 2 reflect the left to right direction of such axioms only.

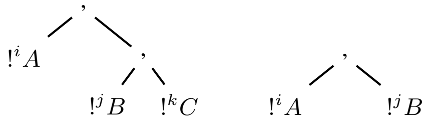
Figure 2 presents the structured system acLL_Σ, for the logic described by the SDML determined by Σ, with $\mathscr{A} = \{\mathsf{C}, \mathsf{W}, \mathsf{A1}, \mathsf{A2}, \mathsf{E}\}$ where, in the subexponential rule for $\mathsf{S} \in \mathscr{A}$, the respective $s \in I$ is such that $\mathsf{S} \in f(s)$ (*e.g.* the subexponential symbol e indicates that $\mathsf{E} \in f(e)$). We will denote by $!^{\mathsf{Ax}}\Delta$ the fact that the structure Δ contains only banged formulas as leaves, each of them assuming the axiom Ax.

As an economic notation, we will write $\uparrow i$ for the *upset* of the index i, *i.e.*, the set $\{j \in I : i \preccurlyeq j\}$. We extend this notation to structures in the following way. Let Γ be a structure containing only banged formulas as leaves. If such formulas admit the multiset partition

$$\{!^j F \in \Gamma : i \preccurlyeq j\} \cup \{!^k F \in \Gamma : i \not\preccurlyeq k \text{ and } \mathsf{W} \in f(k)\}$$

then $\Gamma^{\uparrow i}$ is the structure obtained from Γ by easing the formulas in the second component of the partition (equivalently, the substructure of Γ formed with all and only formulas of the first component of the partition). Otherwise, $\Gamma^{\uparrow i}$ is undefined.

Example 4. Let $\Gamma = (!^i A, (!^j B, !^k C))$ be represented below left, $i \preceq j$ but $i \not\preceq k$, and $\mathsf{W} \in f(k)$. Then $\Gamma^{\uparrow i} = (!^i A, !^j B)$ is depicted below right



Observe that, if $\mathsf{W} \notin f(k)$, then $\Gamma^{\uparrow i}$ cannot be built. In this case, any derivation of $\Gamma \Rightarrow !^i(A \otimes B)$ cannot start with an application of the promotion rule $!^i R$ (similarly to how promotion in ILL cannot be applied in the presence of non-classical contexts). In this case, if A, B are atomic, this sequent would not be provable.

Example 5. The use of subexponentials to deal with associativity can be illustrated by the prefixing sequent $A \to B \Rightarrow (C \to A) \to (C \to B)$: It is not provable for an arbitrary formula C, but if $C = !^a C'$, then

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!^a C' \Rightarrow !^a C'}\,\mathsf{init} \qquad \dfrac{\dfrac{}{A \Rightarrow A}\,\mathsf{init} \quad \dfrac{}{B \Rightarrow B}\,\mathsf{init}}{(A, A \to B) \Rightarrow B}\,{\to} L}{((!^a C', (!^a C' \to A)), (A \to B)) \Rightarrow B}\,{\to} L}{(!^a C', ((!^a C' \to A), (A \to B))) \Rightarrow B}\,\mathsf{A1}}{((!^a C' \to A), (A \to B)) \Rightarrow !^a C' \to B}\,{\to} R}{A \to B \Rightarrow (!^a C' \to A) \to (!^a C' \to B)}\,{\to} R$$

2.1 Cut-Elimination

When it comes to the proof of cut-elimination for acLL_Σ, the cut reductions for the propositional connectives follow the standard steps for similar systems such as, *e.g.*, Moot and Retoré's system $\mathsf{NL}\Diamond$ in [38, Chapter 5.2.2]. The case of structural rules, on the other hand, should be treated with care.

Propositional rules

$$\dfrac{\Gamma\{(F,G)\} \Rightarrow H}{\Gamma\{F \otimes G\} \Rightarrow H}\ \otimes L \qquad \dfrac{\Gamma_1 \Rightarrow F \quad \Gamma_2 \Rightarrow G}{(\Gamma_1,\Gamma_2) \Rightarrow F \otimes G}\ \otimes R \qquad \dfrac{\Gamma\{F\} \Rightarrow H \quad \Gamma\{G\} \Rightarrow H}{\Gamma\{F \oplus G\} \Rightarrow H}\ \oplus L$$

$$\dfrac{\Gamma \Rightarrow F_i}{\Gamma \Rightarrow F_1 \oplus F_2}\ \oplus R_i \qquad \dfrac{\Gamma\{F_i\} \Rightarrow G}{\Gamma\{F_1 \,\&\, F_2\} \Rightarrow G}\ \&L_i \qquad \dfrac{\Gamma \Rightarrow F \quad \Gamma \Rightarrow G}{\Gamma \Rightarrow F \,\&\, G}\ \&R$$

$$\dfrac{\Delta \Rightarrow F \quad \Gamma\{G\} \Rightarrow H}{\Gamma\{(\Delta, F \rightarrow G)\} \Rightarrow H} \rightarrow L \qquad \dfrac{(F,\Gamma) \Rightarrow G}{\Gamma \Rightarrow F \rightarrow G} \rightarrow R \qquad \dfrac{\Delta \Rightarrow F \quad \Gamma\{G\} \Rightarrow H}{\Gamma\{(G \leftarrow F, \Delta)\} \Rightarrow H} \leftarrow L$$

$$\dfrac{(\Gamma,F) \Rightarrow G}{\Gamma \Rightarrow G \leftarrow F} \leftarrow R \qquad \dfrac{\Gamma\{\,\} \Rightarrow F}{\Gamma\{1\} \Rightarrow F}\ 1L \qquad \dfrac{}{\Rightarrow 1}\ 1R \qquad \dfrac{}{\Gamma \Rightarrow \top}\ \top R$$

Initial and cut rules

$$\dfrac{}{F \Rightarrow F}\ \mathsf{init} \qquad \dfrac{\Delta \Rightarrow F \quad \Gamma\left\{{}^1F\right\}\ldots\left\{{}^nF\right\} \Rightarrow G}{\Gamma\left\{{}^1\Delta\right\}\ldots\left\{{}^n\Delta\right\} \Rightarrow G}\ \mathsf{mcut}$$

Fig. 1. Structured system FNL for non-associative, full Lambek calculus.

Subexponential rules

$$\dfrac{\Gamma^{\uparrow i} \Rightarrow F}{\Gamma \Rightarrow\ !^iF}\ !^iR \qquad \dfrac{\Gamma\{F\} \Rightarrow G}{\Gamma\left\{!^iF\right\} \Rightarrow G}\ \mathsf{der}$$

Structural rules

$$\dfrac{\Gamma\{((!^a\Delta_1,\Delta_2),\Delta_3)\} \Rightarrow G}{\Gamma\{(!^a\Delta_1,(\Delta_2,\Delta_3))\} \Rightarrow G}\ \mathsf{A1} \qquad \dfrac{\Gamma\{(\Delta_1,(\Delta_2,!^a\Delta_3))\} \Rightarrow G}{\Gamma\{((\Delta_1,\Delta_2),!^a\Delta_3)\} \Rightarrow G}\ \mathsf{A2} \qquad \dfrac{\Gamma\{(\Delta_2,!^e\Delta_1)\} \Rightarrow G}{\Gamma\{(!^e\Delta_1,\Delta_2)\} \Rightarrow G}\ \mathsf{E1}$$

$$\dfrac{\Gamma\{(!^e\Delta_2,\Delta_1)\} \Rightarrow G}{\Gamma\{(\Delta_1,!^e\Delta_2)\} \Rightarrow G}\ \mathsf{E2} \qquad \dfrac{\Gamma\{\,\} \Rightarrow G}{\Gamma\{!^w\Delta\} \Rightarrow G}\ \mathsf{W} \qquad \dfrac{\Gamma\left\{{}^1!^c\Delta\right\}\ldots\left\{{}^n!^c\Delta\right\} \Rightarrow G}{\Gamma\left\{{}^1\ \right\}\ldots\left\{{}^k!^c\Delta\right\}\ldots\left\{{}^n\ \right\} \Rightarrow G}\ \mathsf{C}$$

Fig. 2. Structured system acLL_Σ for the logic described by Σ.

Theorem 6. *If the sequent $\Gamma \Rightarrow F$ is provable in acLL_Σ, then it has a proof with no instances of the rule* mcut.

Proof. The most representative cases of cut reductions involving subexponentials are detailed next. In order to simplify the notation, when possible, the mcut rule is presented in its simple form, with an 1-ary context.

Case $!^a$: Suppose that

$$\dfrac{\dfrac{\overset{\pi_1}{\Delta_1^{\uparrow a} \Rightarrow F}}{\Delta_1 \Rightarrow\ !^aF}\ !^aR \qquad \dfrac{\overset{\pi_2}{\Gamma\{((!^aF,\Delta_2),\Delta_3)\} \Rightarrow G}}{\Gamma\{(!^aF,(\Delta_2,\Delta_3))\} \Rightarrow G}\ \mathsf{A1}}{\Gamma\{(\Delta_1,(\Delta_2,\Delta_3))\} \Rightarrow G}\ \mathsf{mcut}$$

Since axioms are upwardly closed w.r.t. $\preceq$, it must be the case that $\Delta_1^{\uparrow a}$ contains only formulas marked with subexponentials allowing associativity. All the other formulas in Δ_1 can be weakened; this is guaranteed by the application of the rule $!^a R$ in π_1. Hence the derivation above reduces to

$$\dfrac{\dfrac{\dfrac{\dfrac{\overset{\pi_1}{\Delta_1^{\uparrow a} \Rightarrow F}}{\Delta_1^{\uparrow a} \Rightarrow\, !^a F}\; !^a R \qquad \overset{\pi_2}{\Gamma\{(!^a F, \Delta_2), \Delta_3)\} \Rightarrow G}}{\Gamma\Big\{((\Delta_1^{\uparrow a}, \Delta_2), \Delta_3)\Big\} \Rightarrow G}\; \mathsf{mcut}}{\Gamma\Big\{(\Delta_1^{\uparrow a}, (\Delta_2, \Delta_3))\Big\} \Rightarrow G}\; \mathsf{A1}}{\overline{\Gamma\{(\Delta_1, (\Delta_2, \Delta_3))\} \Rightarrow G}}\; \mathsf{W}$$

Case $!^c$: Suppose that

$$\dfrac{\dfrac{\overset{\pi_1}{\Delta^{\uparrow c} \Rightarrow F}}{\Delta \Rightarrow\, !^c F}\; !^c R \qquad \dfrac{\overset{\pi_2}{\Gamma\{!^c F\} \ldots \{!^c F\} \ldots \{!^c F\} \Rightarrow G}}{\Gamma\{\,\} \ldots \{!^c F\} \ldots \{\,\} \Rightarrow G}\; \mathsf{C}}{\Gamma\{\,\} \ldots \{\Delta\} \ldots \{\,\} \Rightarrow G}\; \mathsf{mcut}$$

Since $\Delta^{\uparrow c}$ contains only formulas marked with subexponentials allowing contraction, the derivation above reduces to

$$\dfrac{\dfrac{\dfrac{\dfrac{\overset{\pi_1}{\Delta^{\uparrow c} \Rightarrow F}}{\Delta^{\uparrow c} \Rightarrow\, !^c F}\; !^c R \qquad \overset{\pi_2}{\Gamma\{!^c F\} \ldots \{!^c F\} \ldots \{!^c F\} \Rightarrow G}}{\Gamma\{\Delta^{\uparrow c}\} \ldots \{\Delta^{\uparrow c}\} \ldots \{\Delta^{\uparrow c}\} \Rightarrow G}\; \mathsf{mcut}}{\Gamma\{\,\} \ldots \{\Delta^{\uparrow c}\} \ldots \{\,\} \Rightarrow G}\; \mathsf{C}}{\overline{\Gamma\{\,\} \ldots \{\Delta\} \ldots \{\,\} \Rightarrow G}}\; \mathsf{W}$$

Observe that here, as usual, the multicut rule is needed in order to reduce the cut complexity.

Case $!^i R$: Suppose that

$$\dfrac{\dfrac{\overset{\pi_1}{\Delta^{\uparrow i} \Rightarrow F}}{\Delta \Rightarrow\, !^i F}\; !^i R \qquad \dfrac{\overset{\pi_2}{\big(\Gamma\{!^i F\}\big)^{\uparrow j} \Rightarrow G}}{\Gamma\{!^i F\} \Rightarrow\, !^j G}\; !^j R}{\Gamma\{\Delta\} \Rightarrow\, !^j G}\; \mathsf{mcut}$$

If $j \not\preceq i$, then it should be the case that $\mathsf{W} \in f(i)$ and $\big(\Gamma\{!^i F\}\big)^{\uparrow j} = \Gamma\{\,\}^{\uparrow j}$, since $!^i F$ will be weakened in the application of rule $!^j R$. Hence, all formulas in Δ can be weakened as well and the reduction is

$$\dfrac{\dfrac{\overset{\pi_2}{\Gamma\{\,\}^{\uparrow j} \Rightarrow G}}{\Gamma\{\,\} \Rightarrow\, !^j G}\; !^j R}{\Gamma\{\Delta\} \Rightarrow\, !^j G}\; \mathsf{W}$$

On the other hand, if $j \preceq i$, by transitivity all the formulas in $\Delta^{\uparrow i}$ also have this property (implying that $\Delta^{\uparrow i}$ is a substructure of $\Delta^{\uparrow j}$), and the rest of formulas of Δ can be weakened. Hence the derivation above reduces to

$$\dfrac{\dfrac{\overset{\pi_1}{\Delta^{\uparrow i} \Rightarrow F}}{\Delta^{\uparrow j} \Rightarrow\, !^i F}\; !^i R \qquad \overset{\pi_2}{\left(\Gamma\{!^i F\}\right)^{\uparrow j} \Rightarrow G}}{\dfrac{(\Gamma\{\Delta\})^{\uparrow j} \Rightarrow G}{\Gamma\{\Delta\} \Rightarrow\, !^j G}\; !^j R}$$

The other cases for subexponentials are similar or simpler. □

The next examples illustrate what we mean by acLL_Σ being a "conservative extension" of subsystems and variants. Indeed, although we remove structural properties of the core LL, subexponentials allow them to be added back, either locally or globally.

Example 7 (Structural variants of iMALL*).* Adding combinations of contraction C and / or weakening W for *arbitrary formulas* to additive-multiplicative intuitionistic linear logic (iMALL) yields, respectively, propositional intuitionistic logic $\mathsf{ILP} = \mathsf{iMALL} + \{\mathsf{C}, \mathsf{W}\}$, and the intuitionistic versions of affine linear logic $\mathsf{aLL} = \mathsf{iMALL} + \mathsf{W}$ and relevant logic $\mathsf{R} = \mathsf{iMALL} + \mathsf{C}$. For the sake of presentation we overload the notation and use the connectives of linear logic also for these logics. In order to embed the logics above into acLL_Σ, let $\alpha \in \{\mathsf{ILP}, \mathsf{aLL}, \mathsf{R}\}$ and consider modalities $!^\alpha$ with $f(\alpha) = \{\mathsf{E}, \mathsf{A1}, \mathsf{A2}\} \cup \mathcal{A}$ where $\mathcal{A} \subseteq \{\mathsf{C}, \mathsf{W}\}$ is the set of axioms whose corresponding rules are in α. The translation τ_α prefixes *every subformula* with the modality $!^\alpha$. For $\mathcal{L} \in \{\mathsf{ILP}, \mathsf{aLL}, \mathsf{R}\}$ it is then straightforward to show that a structured sequent S is cut-free derivable in $\mathcal{L}$ iff its translation $\tau_\alpha(S)$ is cut-free derivable in the logic described by $(\{\alpha\}, \preccurlyeq, f)$ with $\preccurlyeq$ the obvious relation, and f as given above.

Example 8 (Structural variants of FNL*).* Following the same script as above and starting from FNL:

- considering $f(\alpha) = \mathcal{A} \subseteq \{\mathsf{E}, \mathsf{A1}, \mathsf{A2}\}$;
  - If $\mathcal{A} = \{\mathsf{A1}, \mathsf{A2}\}$, then we obtain the system FL;
  - If $\mathcal{A} = \{\mathsf{E}, \mathsf{A1}, \mathsf{A2}\}$ then the resulting system corresponds to iMALL.
  - Adding C, W as options to $\mathcal{A}$ will result the affine/relevant versions of the systems above.
- in a pre-order $(I, \preceq)$, if $f(i) = \{\mathsf{A1}, \mathsf{A2}\} \cup \mathcal{A}_i$ where $\mathcal{A}_i \subseteq \{\mathsf{E}, \mathsf{C}, \mathsf{W}\}$ for each $i \in I$, then the resulting system corresponds to SMALC_Σ in [21] (that is, the extension of FL with subexponentials).

2.2 An Example in Linguistics

Since its inception, Lambek calculus [29] has been applied to the modeling of natural language syntax by means of categorial grammars. In a categorial grammar, each word is assigned one or several Lambek formulas, which serve as syntactic categories. For a simple example, *John* and *Mary* are assigned np ("noun phrase") and *loves* gets

$(np \to s) \leftarrow np$. Here s stands for "sentence", and *loves* is a transitive verb, which lacks noun phrases on both sides to become a sentence. Grammatical validity of *"John loves Mary"* is supported by derivability of the sequent $np, (np \to s) \leftarrow np, np \Rightarrow s$. Notice that this derivability keeps valid also in the non-associative setting, if the correct nested structure is provided: $(np, ((np \to s) \leftarrow np, np)) \Rightarrow s$.

The original Lambek calculus L is associative. In some cases, however, associativity leads to over-generation, *i.e.*, validation of grammatically incorrect sentences. Lambek himself realized this and proposed the non-associative calculus NL in [30]. We will illustrate this issue with the example given in [38, Sect. 4.2.2]. The syntactic category assignment is as follows (where n stands for "noun"):

Words	Types
the	$np \leftarrow n$
Hulk	n
is	$(np \to s) \leftarrow (n \leftarrow n)$
green, *incredible*	$n \leftarrow n$

With this assignment, sentences *"The Hulk is green"* and *"The Hulk is incredible"* are correctly marked as valid, by deriving the sequent

$$(np \leftarrow n, n), ((np \to s) \leftarrow (n \leftarrow n), n \leftarrow n) \Rightarrow s$$

However, in the associative setting the sequent for the phrase *"The Hulk is green incredible,"* which is grammatically incorrect, also becomes derivable:

$$np \leftarrow n, n, (np \to s) \leftarrow (n \leftarrow n), n \leftarrow n, n \leftarrow n \Rightarrow s,$$

essentially due to derivability of $n \leftarrow n, n \leftarrow n \Rightarrow n \leftarrow n$.

In other situations, however, associativity is useful. Standard examples include handling of dependent clauses, *e.g.*, *"the girl whom John loves,"* which is validated as a noun phrase by the following derivable sequent:

$$np \leftarrow n, n, (n \to n) \leftarrow (s \leftarrow np), np, (np \to s) \leftarrow np \Rightarrow np$$

Here $(n \to n) \leftarrow (s \leftarrow np)$ is the syntactic category for *who*.

Our subexponential extension of NL, however, handles this case using local associativity instead of the global one. Namely, the category for *whom* now becomes $(n \to n) \leftarrow (s \leftarrow !^a np)$, where $!^a$ is a subexponential which allows the $\mathsf{A2}$ rule, and the following sequent is happily derivable:

$$np \leftarrow n, (n, ((n \to n) \leftarrow (s \leftarrow !^a np), (np, (np \to s) \leftarrow np))) \Rightarrow np$$

The necessity of this more fine-grained control of associativity, instead of a global associativity rule, is seen via a combination of these examples. Namely, we talk about sentences like *"The superhero whom Hawkeye killed was incredible"* and *"... was green"*. With $!^a$, each of them is handled in the same way as the previous examples:

$$\begin{aligned}&(np \leftarrow n, (n, ((n \to n) \leftarrow (s \leftarrow !^a np), (np, (np \to s) \leftarrow np)))),\\ &\qquad\qquad\qquad\qquad ((np \to s) \leftarrow (n \leftarrow n), n \leftarrow n) \Rightarrow s.\end{aligned}$$

On one hand, without $!^a$ this sequent cannot be derived in the non-associative system. On the other hand, if we make the system globally associative, it would validate incorrect sentences like *"The superhero whom Hawkeye killed was green incredible."*

3 Some Undecidability Results

Non-associativity makes a significant difference in decidability and complexity matters. For example, while L is NP-complete [47], NL is decidable in polynomial time [1,14].

For our system acLL_Σ, its decidability or undecidability depends on its signature Σ. In fact, we have a family of different systems acLL_Σ, with Σ as a parameter. Recall that the subexponential signature Σ controls not just the number of subexponentials and the preorder among them. More importantly, it dictates, for each subexponential, which structural rules this subexponential licenses. If for every $i \in I$ we have $\mathsf{C} \notin f(s)$, that is, no subexponential allows contraction, then acLL_Σ is clearly decidable, since the cut-free proof search space is finite. Therefore, for undecidability it is necessary to have at least one subexponential which allows contraction.

For a non-associative system with only one fully-powered exponential modality s (that is, $f(s) = \{\mathsf{E}, \mathsf{C}, \mathsf{W}, \mathsf{A1}, \mathsf{A2}\}$), undecidability was proven in a preprint by Tanaka [51], based on Chvalovský's [11] result on undecidability of the finitary consequence relation in FNL.

In this section, we prove two undecidability results. The first one is a refinement of Tanaka's result: We establish undecidability with at least one subexponential which allows contraction and weakening (commutativity/associativity are optional), in a subsystem containing only the additive connective $\oplus$ and the multiplicatives $\otimes$ and $\to$.

The second undecidability result is for the minimalistic, purely multiplicative fragment, which includes only $\to$ (not even $\otimes$). As a trade-off, however, it requires two subexponentials: the "main" one, which allows contraction, exchange, and associativity (weakening is optional), and an "auxiliary" one, which allows only associativity.

It should be noted that this undecidability result is orthogonal to Tanaka's [51], and the proof technique is essentially different. Indeed, Chvalovský's undecidability theorem does not hold for the non-associative Lambek calculus without additives, where the consequence relation is decidable [7].

Finally, we observe that *if* the intersection of these systems is decidable (which is still an open question), then our two undecidability results are *incomparable:* we have two undecidable fragments of acLL_Σ, but their common part, which includes only divisions and one exponential, would be decidable.

3.1 Undecidability with Additives and One Subexponential

We are going to derive the next theorem from undecidability of the finitary consequence relation in FNL [11]. Recall that FNL is, in fact, the fragment of acLL_Σ without subexponentials (that is, with an empty I).

Theorem 9. *If there exists such $s \in I$ that $f(s) \supseteq \{\mathsf{C}, \mathsf{W}\}$, then the derivability problem in acLL_Σ is undecidable. Moreover, this holds for the fragment with only $\otimes$, $\to$, $\oplus, !^s$.*

In fact, using C and W, one can also derive $\mathsf{A1}$, $\mathsf{A2}$, $\mathsf{E1}$, and $\mathsf{E2}$. Therefore, if $f(s) \supseteq \{\mathsf{C}, \mathsf{W}\}$, then $!^s$ is actually a full-power exponential modality. (In the proof of Theorem 9 below, we use only W and C rules, in order to avoid confusion.) However, Theorem 9 does not directly follow from undecidability of propositional linear logic [32], because here the basic system is non-associative and non-commutative, while linear logic is both associative and commutative. Thus, we need a different encoding for undecidability.

Let $\mathbf{\Phi}$ be a finite set of FNL sequents. By $\mathsf{FNL}(\mathbf{\Phi})$ let us denote FNL extended by adding sequents from $\mathbf{\Phi}$ as additional (non-logical) axioms. In general, $\mathsf{FNL}(\mathbf{\Phi})$ does not enjoy cut-elimination, so mcut is kept as a rule of inference in $\mathsf{FNL}(\mathbf{\Phi})$. A sequent $\Gamma \Rightarrow F$ is called *a consequence of* $\mathbf{\Phi}$ if this sequent is derivable in $\mathsf{FNL}(\mathbf{\Phi})$.

Theorem 10 (Chvalovský [11]**).** *The consequence relation in* FNL *is undecidable, that is, there exists no algorithm which, given* $\mathbf{\Phi}$ *and* $\Gamma \Rightarrow F$*, determines whether* $\Gamma \Rightarrow F$ *is a consequence of* $\mathbf{\Phi}$*. Moreover, undecidability keeps valid when* $\mathbf{\Phi}$ *and* $\Gamma \Rightarrow F$ *are built from variables using only* $\otimes$ *and* $\oplus$*.*

Now, in order to prove Theorem 9, we internalize $\mathbf{\Phi}$ into the sequent using $!^s$, assuming $f(s) \supseteq \{\mathsf{C}, \mathsf{W}\}$.

First we notice that we may suppose, without loss of generality, that all sequents in $\mathbf{\Phi}$ are of the form $\Rightarrow A$, that is, have empty antecedents. Namely, each sequent of the form $\Pi \Rightarrow B$ can be replaced by $\Rightarrow (\bigotimes \Pi) \to B$, where $\bigotimes \Pi$ is obtained from Π by replacing each comma with $\otimes$. Indeed, these sequents are derivable from one another: from $\Pi \Rightarrow B$ to $\Rightarrow (\bigotimes \Pi) \to B$ we apply a sequence of $\otimes L$ followed by $\to R$, and for the other direction we apply a series of cuts, first with $(\bigotimes \Pi, (\bigotimes \Pi) \to B) \Rightarrow B$, and then with $(F, G) \Rightarrow F \otimes G$ several times, for the corresponding subformulas of $\bigotimes \Pi$. The following embedding lemma ("modalized deduction theorem") holds.

Lemma 11. *The sequent* $\Gamma \Rightarrow F$ *is a consequence of* $\mathbf{\Phi} = \{\Rightarrow A_1, \ldots, \Rightarrow A_n\}$ *if and only if the sequent* $\big((\ldots((!^s A_1, !^s A_2), !^s A_3), \ldots, !^s A_n), \Gamma\big) \Rightarrow F$ *is derivable in* acLL_Σ.

Proof. Let us denote $(\ldots((!^s A_1, !^s A_2), !^s A_3), \ldots, !^s A_n)$ by $!\Phi$. Notice that C and W can be applied to $!\Phi$ as a whole; this is easily proven by induction on n.

For the "only if" direction let us take the derivation of $\Gamma \Rightarrow F$ in $\mathsf{FNL}(\mathbf{\Phi})$ (with cuts) and replace each sequent of the form $\Delta \Rightarrow G$ in it with $(!\Phi, \Delta) \Rightarrow G$, and each sequent of the form $\Rightarrow G$ with $!\Phi \Rightarrow G$. The translations of non-logical axioms from $\mathbf{\Phi}$ are derived as follows:

$$\dfrac{\dfrac{\dfrac{}{A_i \Rightarrow A_i}\ \mathsf{init}}{!^s A_i \Rightarrow A_i}\ \mathsf{der}}{!\Phi \Rightarrow A_i}\ \mathsf{W},\ n-1 \text{ times}$$

Translations of axioms init and $1R$ are derived from the corresponding original axioms by W, n times; $\top R$ remains valid.

Rules $\otimes L$, $\oplus L$, $\oplus R_i$, $\& L_i$, $\& R$, and $1L$ remain valid. For $\to L$, $\leftarrow L$, and mcut we contract $!\varPhi$ as a whole:

$$\dfrac{\dfrac{(!\varPhi,\varDelta)\Rightarrow F \quad (!\varPhi,\varGamma\{G\})\Rightarrow H}{(!\varPhi,\varGamma\{((!\varPhi,\varDelta),F\to G)\})\Rightarrow H}\;{\to L}}{(!\varPhi,\varGamma\{(\varDelta,F\to G)\})\Rightarrow H}\;\mathsf{C} \qquad \dfrac{\dfrac{(!\varPhi,\varDelta)\Rightarrow F \quad (!\varPhi,\varGamma\{F\}\dots\{F\})\Rightarrow C}{(!\varPhi,\varGamma\{(!\varPhi,\varDelta)\}\dots\{(!\varPhi,\varDelta)\})\Rightarrow C}\;\mathsf{mcut}}{(!\varPhi,\varGamma\{\varDelta\}\dots\{\varDelta\})\Rightarrow C}\;\mathsf{C}$$

For $\otimes R$, $\to R$, and $\leftarrow R$, we combine contraction and weakening:

$$\dfrac{\dfrac{\dfrac{(!\varPhi,\varGamma_1)\Rightarrow F \quad (!\varPhi,\varGamma_2)\Rightarrow G}{((!\varPhi,\varGamma_1),(!\varPhi,\varGamma_2))\Rightarrow F\otimes G}\;{\otimes R}}{(!\varPhi,((!\varPhi,\varGamma_1),(!\varPhi,\varGamma_2)))\Rightarrow F\otimes G}\;\mathsf{W}}{(!\varPhi,(\varGamma_1,\varGamma_2))\Rightarrow F\otimes G}\;\mathsf{C} \qquad \dfrac{\dfrac{\dfrac{(!\varPhi,(F,\varGamma))\Rightarrow G}{(!\varPhi,(F,(!\varPhi,\varGamma))\Rightarrow G}\;\mathsf{W}}{(F,(!\varPhi,\varGamma))\Rightarrow G}\;\mathsf{C}}{(!\varPhi,\varGamma)\Rightarrow F\to G}\;{\to R}$$

Notice that our original derivation was in $\mathsf{FNL}(\mathbf{\Phi})$, so it does not include rules operating subexponentials.

For the "if" direction we take a cut-free proof of $(!\varPhi,\varGamma)\Rightarrow F$ in acLL_Σ and erase all formulas which include the subexponential. In the resulting derivation tree all rules and axioms, except those which operate $!^s$, remain valid. Structural rules for $!^s$ trivialize (since the !-formula was erased). The $!^s R$ rule could not have been used, since we do not have positive occurrences of $!^s F$, and our proof is cut-free.

Finally, der translates into

$$\dfrac{\varGamma\{A_i\}\Rightarrow G}{\varGamma\{\}\Rightarrow G}$$

This is modeled by cut with one of the sequents from $\mathbf{\Phi}$:

$$\dfrac{\Rightarrow A_i \quad \varGamma\{A_i\}\Rightarrow G}{\varGamma\{\}\Rightarrow G}\;\mathsf{mcut}$$

Thus, we get a correct derivation in $\mathsf{FNL}(\mathbf{\Phi})$. □

Theorem 10 and Lemma 11 immediately yield Theorem 9.

3.2 Undecidability Without Additives and with Two Subexponentials

Theorem 12. *If there are $a, c \in I$ such that $f(a) = \{\mathsf{A1}, \mathsf{A2}\}$ and $f(c) \supseteq \{\mathsf{C}, \mathsf{E}, \mathsf{A1}, \mathsf{A2}\}$, then the derivability problem in acLL_Σ is undecidable. Moreover, this holds for the fragment with only $\to$, $!^a$, and $!^c$.*

Remember from Example 8 that SMALC_Σ [21] denotes the extension of FL with subexponentials. The undecidability theorem above is proved by encoding the one-division fragment of SMALC_Σ containing one exponential c such that $f(c) \supseteq \{\mathsf{C}, \mathsf{E}\}$. It turns out that that such a system is undecidable.

Theorem 13 (Kanovich et al. [22,23]). *If there exists such $c \in I$ that $f(c) \supseteq \{\mathsf{C}, \mathsf{E}\}$, then the derivability problem in SMALC_Σ is undecidable. Moreover, this holds for the fragment with only $\to$ and $!^c$.*

Observe that SMALC_Σ can be obtained from acLL_Σ by adding "global" associativity rules:

$$\frac{\Gamma\{((\Delta_1,\Delta_2),\Delta_3)\} \Rightarrow G}{\Gamma\{(\Delta_1,(\Delta_2,\Delta_3))\} \Rightarrow G} \qquad \frac{\Gamma\{(\Delta_1,(\Delta_2,\Delta_3))\} \Rightarrow G}{\Gamma\{((\Delta_1,\Delta_2),\Delta_3)\} \Rightarrow G}$$

The usual formulation of SMALC_Σ, of course, uses sequences of formulas instead of nested structures as antecedents. The alternative formulation, however, would be more convenient for us now. It will be also convenient for us to regard all subexponentials in SMALC_Σ to be associative, that is, $f(s) \supseteq \{\mathsf{A1},\mathsf{A2}\}$ for each $s \in I$.

In order to embed SMALC_Σ into acLL_Σ, we define two translations, $A^{!-}$ and $A^{!+}$, by mutual recursion:

$$\begin{array}{lll}
z^{!-} = {!^a} z & z^{!+} = z & \text{where } z \text{ is a variable,1, or } \top \\
(A \to B)^{!-} = {!^a}(A^{!+} \to B^{!-}) & (A \to B)^{!+} = A^{!-} \to B^{!+} & \\
(B \leftarrow A)^{!-} = {!^a}(B^{!-} \leftarrow A^{!+}) & (B \leftarrow A)^{!+} = B^{!+} \leftarrow A^{!-} & \\
(A \circledast B)^{!-} = {!^a}(A^{!-} \circledast B^{!-}) & (A \circledast B)^{!+} = A^{!+} \circledast B^{!+} & \text{where } \circledast \in \{\otimes,\oplus,\&\} \\
({!^s}A)^{!-} = {!^s}(A^{!-}) & ({!^s}A)^{!+} = {!^s}(A^{!+}) &
\end{array}$$

Informally, our translation adds a $!^a$ over any formula (not only over atoms) of negative polarity, unless this formula was already marked with a $!^s$. Thus, all formulae in antecedents would begin with either the new subexponential $!^a$ or one of the old subexponentials $!^s$, and all these subexponentials allow associativity rules A1 and A2.

Lemma 14. *A sequent $A_1,\ldots,A_n \Rightarrow B$ is derivable in* SMALC_Σ *if and only if its translation $(\ldots(A_1^{!-},A_2^{!-}),\ldots,A_n^{!-}) \Rightarrow B^{!+}$ is derivable in* acLL_Σ.

Proof. For the "only if" part, let us first note that each formula $A_i^{!-}$ is of the form $!^s F$ and $\mathsf{A1},\mathsf{A2} \in f(s)$. Indeed, either s is an "old" subexponential label (for which we added $\mathsf{A1},\mathsf{A2}$) or $s = a$. Thus brackets can be freely rearranged in the antecedent.

Now we take a cut-free proof of $A_1,\ldots,A_n \Rightarrow B$ in SMALC_Σ and replace each sequent in it with its translation. Right rules for connectives other than subexponentials, i.e., $\otimes R$, $\oplus R_i$, $\& R$, $\to R$, and $\leftarrow R$, remain valid as they are, up to rearranging brackets in antecedents. For $!^i R$, we notice that the translation of a formula of the form $!^j F$, where $j \preceq i$, is also a formula of the form $!^j F'$. Thus, this rule also remains valid. The same holds for the dereliction rule der, because $(!^i F)^{!-}$ is exactly $!^i(F^{!-})$. Finally, the "old" structural rules (exchange, contraction, weakening) also remain valid (up to rearranging of brackets), since $!^i F$ gets translated into $!^i(F^{!-})$, which enjoys the same structural rules.

For the other left rules, we need to derelict $!^a$ first, and then perform the corresponding rule application. Rearrangement of brackets, if needed, is performed below dereliction or above the application of the rule in question.

The "if" part is easier. Given a derivation of $(\ldots(A_1^{!-},A_2^{!-}),\ldots,A_n^{!-}) \Rightarrow B^{!+}$ in acLL_Σ, we erase $!^a$ everywhere, and consider it as a derivation in SMALC_Σ. Associativity rules for the erased $!^a$ (which are the only structural rules for this subexponential) keep valid, because now associativity is global. Dereliction and right introduction for $!^a$ trivialize. All other rules, which do not operate $!^a$, remain as they are. Thus, we get a derivation of $A_1,\ldots,A_n \Rightarrow B$ in SMALC_Σ, since erasing $!^a$ makes our translations just identical. □

4 Related Work and Conclusion

In this paper, we have presented acLL_Σ, a sequent-based system for non-associative, non-commutative linear logic with subexponentials. Starting form FNL, we modularly and uniformly added rules for exchange, associativity, weakening and contraction, which can be applied with the subexponentials having with the respective features. This allows for the application of structural rules locally, and it conservatively extends well known systems in the literature, continuing the path of controlling structural properties started by Girard himself [16].

Another approach to combining associative and non-associative behavior in Lambek-style grammars is the framework of *the Lambek calculus with brackets* by Morrill [39,40] and Moortgat [34]. The bracket approach is dual to ours: there the base system is associative, and brackets, which are controlled by bracket modalities, introduce local non-associativity. Both the associative Lambek calculus and the non-associative Lambek calculus can be embedded into the Lambek calculus with brackets: the former is just by design of the system and the latter was shown by Kurtonina [26] by constructing a translation.

From the point of view of generative power, however, the (associative) Lambek calculus with brackets is weaker than the non-associative system with subexponentials, which is presented in this paper. Namely, as shown by Kanazawa [19], grammars based on the Lambek calculus with brackets can generate only context-free languages. In contrast, grammars based on our system with subexponentials go beyond context-free languages, even when no subexponential allows contraction (subexponentials allowing contraction may lead to undecidability, as shown in the last section).

As a quick example, let us consider a subexponential $!^{ae}$ which allows both associativity (A1 and A2) and exchange (E). If we put this subexponential over any (sub)formula, the system becomes associative and commutative. Using this system, one can describe the non context-free language MIX_3, which contains all non-empty words over $\{a, b, c\}$, in which the numbers of a, b, and c are equal. Indeed, MIX_3 is the permutation closure of the language $\{(abc)^n \mid n \geq 1\}$. The latter is regular, therefore context-free, and therefore definable by a Lambek grammar. The ability of our system to go beyond context-free languages is important from the point of view of applications, since there are known linguistic phenomena which are essentially non-context-free [49].

Regarding decidability, let us compare our results with the more well-known associative non-commutative and associative commutative cases.

In the associative and commutative case the situation is as follows. In the presence of additives, the system is known to be undecidable with one exponential modality [32]. Without additives, we get MELL, the (un)decidability of which is a well-known open problem [50]. However, with two subexponentials MELL again becomes undecidable [9]. Thus, we have the same trade-off as in our non-associative non-commutative case: for undecidability one needs either additives, or two subexponentials.

Our results help to shed some light in the (un)decidability problem for the spectrum of logical systems surrounding MELL/FNL, allowing for a fine-grained analysis of the problem, specially the trade-offs on connectives and subexponentials for guaranteeing (un)decidability.

There is a lot to be done from now on. First of all, we would like to analyze better the minimalist fragment of acLL_Σ containing only implication and one fully-powered subexponential, as it seems to be crucial for understanding the lower bound of undecidability (or the upper bound of decidability). Second, one should definitely explore more the use of acLL_Σ in modeling natural language syntax. The examples in Sect. 2.2 show how to locally combine sentences with different grammatical characteristics, and the MIX_3 example above illustrates how that can be of importance. That is, it would be interesting to have a formal study about acLL_Σ and categorial grammars. Third, we plan to investigate the connections between our work and Adjoint logic [48] as well as with Display calculus [5,12]. Finally, we intend to study proof-theoretic properties of acLL_Σ, such as normalization of proofs (*e.g.* via focusing) and interpolation.

Acknowledgements. We are grateful for the useful suggestions from the anonymous referees. We would like to thank L. Beklemishev, M. Moortgat, and C. Retoré for their inspiring and helpful comments regarding an approach based on non-associativity.

References

1. Aarts, E., Trautwein, K.: Non-associative Lambek categorial grammar in polynomial time. Math. Log. Q. **41**(4), 476–484 (1995)
2. Abrusci, V.M.: A comparison between Lambek syntactic calculus and intuitionistic linear logic. Zeitschr. Math. Logik Grundl. Math. (Math. Logic Q.) **36**, 11–15 (1990)
3. Ajdukiewicz, K.: Die syntaktische Konnexität. Studia Philosophica **1**, 1–27 (1935)
4. Bar-Hillel, Y.: A quasi-arithmetical notation for syntactic description. Language **29**, 47–58 (1953)
5. Belnap, N.: Display logic. J. Philos. Log. **11**(4), 375–417 (1982). https://doi.org/10.1007/BF00284976
6. Bulinska, M.: On the complexity of nonassociative Lambek calculus with unit. Stud. Logica **93**(1), 1–14 (2009). https://doi.org/10.1007/s11225-009-9205-2
7. Buszkowski, W.: Lambek calculus with nonlogical axioms. In: Language and Grammar, Studies in Mathematical Linguistics and Natural Language, pp. 77–93. CSLI Publications (2005)
8. Buszkowski, W., Farulewski, M.: Nonassociative Lambek calculus with additives and context-free languages. In: Grumberg, O., Kaminski, M., Katz, S., Wintner, S. (eds.) Languages: From Formal to Natural. LNCS, vol. 5533, pp. 45–58. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01748-3_4
9. Chaudhuri, K.: Undecidability of multiplicative subexponential logic. In: Alves, S., Cervesato, I. (eds.) Proceedings Third International Workshop on Linearity, LINEARITY 2014, Vienna, Austria, 13th July 2014. EPTCS, vol. 176, pp. 1–8 (2014). https://doi.org/10.4204/EPTCS.176.1
10. Chaudhuri, K., Marin, S., Straßburger, L.: Modular focused proof systems for intuitionistic modal logics. In: FSCD, pp. 16:1–16:18 (2016)
11. Chvalovský, K.: Undecidability of consequence relation in full non-associative Lambek calculus. J. Symb. Logic **80**(2), 567–586 (2015)
12. Clouston, R., Dawson, J., Goré, R., Tiu, A.: Annotation-free sequent calculi for full intuitionistic linear logic. In: Rocca, S.R.D. (ed.) Computer Science Logic 2013 (CSL 2013), CSL 2013, Torino, Italy, 2–5 September 2013. LIPIcs, vol. 23, pp. 197–214. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013). https://doi.org/10.4230/LIPIcs.CSL.2013.197

13. Danos, V., Joinet, J.-B., Schellinx, H.: The structure of exponentials: uncovering the dynamics of linear logic proofs. In: Gottlob, G., Leitsch, A., Mundici, D. (eds.) KGC 1993. LNCS, vol. 713, pp. 159–171. Springer, Heidelberg (1993). https://doi.org/10.1007/BFb0022564
14. de Groote, P., Lamarche, F.: Classical non-associative Lambek calculus. Stud. Logica **71**(3), 355–388 (2002). https://doi.org/10.1023/A:1020520915016
15. Gheorghiu, A., Marin, S.: Focused proof-search in the logic of bunched implications. In: FOSSACS 2021. LNCS, vol. 12650, pp. 247–267. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-71995-1_13
16. Girard, J.-Y.: Linear logic. Theor. Comput. Sci. **50**, 1–102 (1987). https://doi.org/10.1016/0304-3975(87)90045-4
17. Guglielmi, A., Straßburger, L.: Non-commutativity and MELL in the calculus of structures. In: Fribourg, L. (ed.) CSL 2001. LNCS, vol. 2142, pp. 54–68. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44802-0_5
18. Hepple, M.: A general framework for hybrid substructural categorial logics. Technical report 94–14, IRCS (1994)
19. Kanazawa, M.: On the recognizing power of the Lambek calculus with brackets. J. Logic Lang. Inform. **27**(4), 295–312 (2018)
20. Kanovich, M., Kuznetsov, S., Nigam, V., Scedrov, A.: A logical framework with commutative and non-commutative subexponentials. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS (LNAI), vol. 10900, pp. 228–245. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94205-6_16
21. Kanovich, M., Kuznetsov, S., Nigam, V., Scedrov, A.: Subexponentials in non-commutative linear logic. Math. Struct. Comput. Sci. **29**(8), 1217–1249 (2019). https://doi.org/10.1017/S0960129518000117
22. Kanovich, M., Kuznetsov, S., Scedrov, A.: Undecidability of the Lambek calculus with a relevant modality. In: Foret, A., Morrill, G., Muskens, R., Osswald, R., Pogodalla, S. (eds.) FG 2015-2016. LNCS, vol. 9804, pp. 240–256. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53042-9_14
23. Kanovich, M., Kuznetsov, S., Scedrov, A.: The multiplicative-additive Lambek calculus with subexponential and bracket modalities. J. Log. Lang. Inform. **30**, 31–88 (2021)
24. Kopylov, A.: Decidability of linear affine logic. Inf. Comput. **164**(1), 173–198 (2001). https://doi.org/10.1006/inco.1999.2834
25. Kozak, M.: Cyclic involutive distributive full Lambek calculus is decidable. J. Log. Comput. **21**(2), 231–252 (2011). https://doi.org/10.1093/logcom/exq021
26. Kurtonina, N.: Frames and labels. A modal analysis of categorial inference. Ph.D. thesis, Universiteit Utrecht, ILLC (1995)
27. Kurtonina, N., Moortgat, M.: Structural control. In: Blackburn, P., de Rijke, M. (eds.) Specifying Syntactic Structures, CSLI, Stanford, pp. 75–113 (1997)
28. Lamarche, F.: On the Algebra of Structural Contexts. Mathematical Structures in Computer Science, p. 51 (2003). Article dans revue scientifique avec comité de lecture. https://hal.inria.fr/inria-00099461
29. Lambek, J.: The mathematics of sentence structure. Am. Math. Monthly **65**(3), 154–170 (1958)
30. Lambek, J.: On the calculus of syntactic types. In: Jakobson, R. (ed.) Structure of Language and Its Mathematical Aspects, pp. 166–178. American Mathematical Society (1961)
31. Lellmann, B., Olarte, C., Pimentel, E.: A uniform framework for substructural logics with modalities. In: LPAR-21, pp. 435–455 (2017)
32. Lincoln, P., Mitchell, J., Scedrov, A., Shankar, N.: Decision problems for propositional linear logic. Ann. Pure Appl. Logic **56**(1–3), 239–311 (1992)
33. Moortgat, M.: Multimodal linguistic inference. Log. J. IGPL **3**(2–3), 371–401 (1995). https://doi.org/10.1093/jigpal/3.2-3.371

34. Moortgat, M.: Multimodal linguistic inference. J. Logic Lang. Inform. **5**(3–4), 349–385 (1996)
35. Moortgat, M., Morrill, G.: Heads and phrases: type calculus for dependency and constituent structure. Technical report (1991)
36. Moortgat, M., Oehrle, R.: Logical parameters and linguistic variation. In: Fifth European Summer School in Logic, Language and Information. Lecture Notes on Categorial Grammar (1993)
37. Moot, R.: The grail theorem prover: type theory for syntax and semantics. CoRR, abs/1602.00812 (2016). arXiv:1602.00812
38. Moot, R., Retoré, C.: The Logic of Categorial Grammars. LNCS, vol. 6850. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31555-8
39. Morrill, G.: Categorial formalisation of relativisation: Pied piping, islands, and extraction sites. Technical report LSI-92-23-R, Universitat Politècnica de Catalunya (1992)
40. Morrill, G.: Parsing/theorem-proving for logical grammar CatLog3. J. Log. Lang. Inf. **28**(2), 183–216 (2019). https://doi.org/10.1007/s10849-018-09277-w
41. Morrill, G., Leslie, N., Hepple, M., Barry, G.: Categorial deductions and structural operations. In: Studies in Categorial Grammar, Edinburgh Working Paper in Cognitive Science, vol. 5, pp. 1–21 (1990)
42. Nigam, V., Miller, D.: A framework for proof systems. J. Autom. Reason. **45**(2), 157–188 (2010). https://doi.org/10.1007/s10817-010-9182-1
43. Nigam, V., Olarte, C., Pimentel, E.: On subexponentials, focusing and modalities in concurrent systems. Theor. Comput. Sci. **693**, 35–58 (2017). https://doi.org/10.1016/j.tcs.2017.06.009
44. Nigam, V., Pimentel, E., Reis, G.: An extended framework for specifying and reasoning about proof systems. J. Log. Comput. **26**(2), 539–576 (2016). https://doi.org/10.1093/logcom/exu029
45. Oehrle, R., Zhang, S.: Lambek calculus and preposing of embedded subjects. Coyote Papers (1989). http://hdl.handle.net/10150/226572
46. Olarte, C., Chiarugi, D., Falaschi, M., Hermith, D.: A proof theoretic view of spatial and temporal dependencies in biochemical systems. Theor. Comput. Sci. **641**, 25–42 (2016). https://doi.org/10.1016/j.tcs.2016.03.029
47. Pentus, M.: Lambek calculus is NP-complete. Theor. Comput. Sci. **357**, 186–201 (2006)
48. Pruiksma, K., Chargin, W., Pfenning, F., Reed, J.: Adjoint logic (2018, Unpublished manuscript)
49. Shieber, S.: Evidence against the context-freeness of natural languages. Linguist. Philos. **8**, 333–343 (1985)
50. Straßburger, L.: On the decision problem for MELL. Theor. Comput. Sci. **768**, 91–98 (2019). https://doi.org/10.1016/j.tcs.2019.02.022
51. Tanaka, H.: A note on undecidability of propositional non-associative linear logics (2019). arXiv preprint arXiv:1909.13444

Effective Semantics for the Modal Logics K and KT via Non-deterministic Matrices

Ori Lahav[1] and Yoni Zohar[2](✉)

[1] Tel Aviv University, Tel Aviv-Yafo, Israel
[2] Bar Ilan University, Ramat Gan, Israel
yoni.zohar@biu.ac.il

Abstract. A four-valued semantics for the modal logic K is introduced. Possible worlds are replaced by a hierarchy of four-valued valuations, where the valuations of the first level correspond to valuations that are legal w.r.t. a basic non-deterministic matrix, and each level further restricts its set of valuations. The semantics is proven to be effective, and to precisely capture derivations in a sequent calculus for K of a certain form. Similar results are then obtained for the modal logic KT, by simply deleting one of the truth values.

1 Introduction

Propositional modal logics extend classical logic with *modalities*, intuitively interpreted as necessity, knowledge, or temporal operators. Such extensions have several applications in computer science and artificial intelligence (see, e.g., [7,9,13]).

The most common and successful semantic framework for modal logics is the so called *possible worlds semantics*, in which each world is equipped with a two-valued valuation, and the semantic constraints regarding the modal operators consider the valuations in *accessible* worlds. While this has been the gold standard for modal logic semantics for many years, alternative semantic frameworks have been proposed. One of these approaches, initiated by Kearns [10], is based on an infinite sequence of sets of valuations in a non-deterministic many-valued semantics. Since then, several non-deterministic many-valued semantics, without possible worlds, were developed for modal logics (see, e.g., [4,8,12,14]). The current paper is a part of that body of work. Having an alternative semantic framework for modal logics, different than the common possible worlds semantics, has the potential of exposing new intuitions and understandings of modal logics, and also to form the basis to new decision procedures.

Our main contribution is a four-valued semantics for the modal logic K. The key characteristic of the semantics that we present is *effectiveness*: when checking

We thank the anonymous reviewers for their useful feedback. This research was supported by NSF-BSF (grant number 2020704), ISF (grant numbers 619/21 and 1566/18), and the Alon Young Faculty Fellowship.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 468–485, 2022.
https://doi.org/10.1007/978-3-031-10769-6_28

for the entailment of a formula φ from a set Γ of formulas in K, it suffices to only consider *partial* models, defined over the subformulas of Γ and φ. To the best of our knowledge, this is the first effective Nmatrices-based semantics for K. Such a semantics has the potential of being subject to reductions to classical satisfiability [3], as it is based on finite-valued truth tables, and thus improving the performance of solvers for modal logic by utilizing off-the-shelf SAT solvers. Another advantage of this semantics is that it precisely captures derivations in a sequent calculus for K that admit a certain property. Following Kearns, models of this semantics are based on the concept of *levels*—valuations of level 0 are the ordinary valuations of Nmatrices, while each level $m > 0$ introduces more constraints. We show that valuations of level m correspond to derivations in the calculus whose largest number of applications of the rule that correspond to the axiom (K) in any branch of the derivation is at most m. Our restrictions between the levels are more complex than the original restrictions in Kearns' work, in order to obtain effectiveness. Another precise correspondence between the semantics and the proof system that we prove, is between the domains of valuations and the formulas allowed to be used in derivations.

Finally, we observe that by deleting one of the truth values, a three-valued semantics for the modal logic KT is obtained, which is similar to the one presented in [8]. Like the case of K, the resulting semantics is effective, and tightly correspond to derivations in a sequent calculus for KT.

Outline. The paper is organized as follows: Sect. 2 reviews standard notions in non-deterministic matrices. In Sect. 3, we present our semantics for the modal logic K, as well as the sequent calculus our investigation will be based on, which is coupled with the notion of (K)-depth of derivations. In Sect. 4, we prove soundness and completeness theorems between the sequent calculus and the semantics. In Sect. 5, we prove that the semantics that we provide is effective, not only for deciding entailment, but also for producing countermodels when an entailment does not hold. In Sect. 6 we establish similar results for the modal logic KT. We conclude with §7, where directions for future research are outlined.

Related Work. In [10], Kearns initiated the study of modal semantics without possible worlds. This work was recently revisited by Skurt and Omori [14], who generalized Kearns' work and reframed his framework within the framework of logical Non-deterministic matrices. As indicated in [14], it was not clear how to make this semantics effective, as it requires checking truth values of infinitely many formulas when considering the validity of a given formula (see, e.g., Remark 42 of [14]). In [4], Coniglio et al. develop a similar framework for modal logics, and some bound over the formulas that need to be considered was achieved. However, in [5], the authors clarified that it is unclear how to effectively use the resulting semantics. A semantics based on Nmatrices for the modal logics KT and S4 was presented in [8] by Grätz, that includes a method to extend a partial model in that semantics into a total one, which results in an effective semantics. We chose here to focus on K, which is a weaker logic, forming a common basis to all other normal modal logics. By deleting one out of four truth values, we obtain corresponding results for KT as well. The semantics that we present here is similar in nature to the one presented in [8], however: (*i*) the truth tables are different, as we intentionally enforced the many-valued tables of the classical connectives to be obtained by a straightforward duplication of truth values from the original two-valued truth tables; and (*ii*) the semantic condition for levels of valuations that we define here is inductive, where each level relies on lower levels (thus refraining from a definition of a more cyclic nature as the one in [8], that is better understood operationally). A variant of the semantics from [14] was also introduced and studied in [12], but without considering the ability to perform effective automated reasoning but instead focusing on infinite valuations rather than on partial ones. A complete proof theoretic characterization in terms of sequent calculi to the various levels of valuations was not given in any of the above works. Also, an effective semantics for K, which is the most basic modal logic, was not given in any of the above works.

Non-deterministic matrices were introduced in [2], and have since became a useful tool for investigating non-classical logics and proof systems (see [1] for a survey). They generalize (deterministic) matrices [15] by allowing a non-deterministic choice of truth values in the truth tables. Like matrices, Nmatrices enjoy the semantic *analyticity* property, which allows one to extend a partial valuation into a full one. Our semantic framework can be viewed as a further refinement of non-deterministic matrices, namely *restricted* non-deterministic matrices, introduced in [6].

2 Preliminaries

In this section we provide the necessary definitions about Nmatrices following [1]. We assume a propositional language $\mathcal{L}$ with countably infinitely many atomic variables $p_1, p_2, \ldots$. When there is no room for confusion, we identify $\mathcal{L}$ with its set of well-formed formulas (e.g., when writing $\varphi \in \mathcal{L}$). We write $sub(\varphi)$ for the set of subformulas of a formula φ. This notation is extended to sets of formulas in the natural way.

Valuations. In the context of a set $\mathcal{V}$ of "truth values", a *valuation* is a function v from some domain $\mathsf{Dom}(v) \subseteq \mathcal{L}$ to $\mathcal{V}$. For a set $\mathcal{F} \subseteq \mathcal{L}$, an *$\mathcal{F}$-valuation* is a valuation with domain $\mathcal{F}$. (In particular, an $\mathcal{L}$-valuation is defined on all formulas.) For $X \subseteq \mathcal{V}$, we write $v^{-1}[X]$ for the set $\{\varphi \mid v(\varphi) \in X\}$. For $x \in \mathcal{V}$, we also write $v^{-1}[x]$ for the set $\{\varphi \mid v(\varphi) = x\}$.

Definition 1. Let $\mathcal{D} \subseteq \mathcal{V}$ be a set of "designated truth values". A valuation v *$\mathcal{D}$-satisfies* a formula φ, denoted by $v \models_{\mathcal{D}} \varphi$, if $v(\varphi) \in \mathcal{D}$. For a set Σ of formulas, we write $v \models_{\mathcal{D}} \Sigma$ if $v \models_{\mathcal{D}} \varphi$ for every $\varphi \in \Sigma$.

Notation 2. Let $\mathcal{D} \subseteq \mathcal{V}$ be a set of designated truth values and $\mathbb{V}$ be a set of valuations. For sets L, R of formulas, we write $L \vdash^{\mathbb{V}}_{\mathcal{D}} R$ if for every $v \in \mathbb{V}$, $v \models_{\mathcal{D}} L$ implies that $v \models_{\mathcal{D}} \varphi$ for some $\varphi \in R$. We omit L or R in this notation when they are empty (e.g., when writing $\vdash^{\mathbb{V}}_{\mathcal{D}} R$), and set parentheses for singletons (e.g., when writing $L \vdash^{\mathbb{V}}_{\mathcal{D}} \varphi$).

Nmatrices. An Nmatrix M for $\mathcal{L}$ is a triple of the form $\langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$, where $\mathcal{V}$ is a set of *truth values*, $\mathcal{D} \subseteq \mathcal{V}$ is a set of *designated truth values*, and $\mathcal{O}$ is a function assigning a *truth table* $\mathcal{V}^n \to P(\mathcal{V}) \setminus \{\emptyset\}$ to every n-ary connective $\diamond$ of $\mathcal{L}$ (which assigns a set of possible values to each tuple of values). In the context of an Nmatrix $M = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$, we often denote $\mathcal{O}(\diamond)$ by $\tilde{\diamond}$.

An $\mathcal{F}$-valuation v is *M-legal* if $v(\varphi) \in \mathsf{pos\text{-}val}(\varphi, M, v)$ for every formula $\varphi \in \mathcal{F}$ whose immediate subformulas are contained in $\mathcal{F}$, where $\mathsf{pos\text{-}val}(\varphi, M, v)$ is defined by:

1. $\mathsf{pos\text{-}val}(p, M, v) = \mathcal{V}$ for every atomic formula p.
2. $\mathsf{pos\text{-}val}(\diamond(\psi_1, \ldots, \psi_n), M, v) = \tilde{\diamond}(v(\psi_1), \ldots, v(\psi_n))$ for every non-atomic formula $\diamond(\psi_1, \ldots, \psi_n)$.

In other words, there is no restriction regarding the values assigned to atomic formulas, whereas the values of compound formulas should respect the truth tables.

Lemma 1 ([1]). *Let $\mathcal{F} \subseteq \mathcal{L}$ be a set closed under subformulas and M an Nmatrix for $\mathcal{L}$. Then every M-legal $\mathcal{F}$-valuation v can be extended to an M-legal $\mathcal{L}$-valuation.*

3 The Modal Logic K

In this section we introduce a novel effective semantics for the model logic K. We first present a known proof system for this logic (Sect. 3.1), and then our semantics (Sect. 3.2). From here on, we assume that the language $\mathcal{L}$ consists of the connectives $\supset$, $\wedge$, $\vee$, $\neg$ and $\Box$ with their usual arities. The standard $\Diamond$ operator can be defined as a macro $\Diamond\varphi \overset{\text{def}}{=} \neg\Box\neg\varphi$. Obviously, using De-Morgan rules, fewer connectives can be used. However, we chose this set of connectives in order to have a primitive language rich enough for the examples that we include along the paper.

3.1 Proof System

Figure 1 presents a Gentzen-style calculus, denoted by $\mathsf{G_K}$, for the modal logic K that was proven to be equivalent to the original formulation of the logic as a Hilbert system (see, e.g., [16]). We take *sequents* to be pairs $\langle \Gamma, \Delta \rangle$ of finite *sets* of formulas. For readability, we write $\Gamma \Rightarrow \Delta$ instead of $\langle \Gamma, \Delta \rangle$ and use standard notations such as $\Gamma, \varphi \Rightarrow \psi$ instead of $(\Gamma \cup \{\varphi\}) \Rightarrow \{\psi\}$.

The (CUT) rule is included in $\mathsf{G_K}$ for convenience, but applications of (CUT) can be eliminated from derivations (see, e.g., [11]). Since the focus of this paper is semantics rather than cut-elimination, we allow ourselves to use cut freely and do not distinguish derivations that use it from derivations that do not. We write $\vdash_{\mathsf{G_K}} \Gamma \Rightarrow \Delta$ if there is a derivation of a sequent $\Gamma \Rightarrow \Delta$ in the calculus $\mathsf{G_K}$.

In the sequel, we provide a semantic characterization of $\vdash_{\mathsf{G_K}}$. It is based on a more refined notion of derivability that takes into account: (i) the set $\mathcal{F}$ of formulas used in the derivation; and (ii) the (K)-*depth* of the derivation, as defined next.

$$(\textsc{weak})\ \frac{\Gamma \Rightarrow \Delta}{\Gamma, \Gamma' \Rightarrow \Delta, \Delta'} \qquad (\textsc{id})\ \frac{}{\Gamma, \varphi \Rightarrow \varphi, \Delta} \qquad (\textsc{cut})\ \frac{\Gamma, \varphi \Rightarrow \Delta \quad \Gamma \Rightarrow \varphi, \Delta}{\Gamma \Rightarrow \Delta} \qquad (K)\ \frac{\Gamma \Rightarrow \varphi}{\Box\Gamma \Rightarrow \Box\varphi}$$

$$(\neg \Rightarrow)\ \frac{\Gamma \Rightarrow \varphi, \Delta}{\Gamma, \neg\varphi \Rightarrow \Delta} \qquad (\Rightarrow \neg)\ \frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \neg\varphi, \Delta} \qquad (\supset\Rightarrow)\ \frac{\Gamma \Rightarrow \varphi, \Delta \quad \Gamma, \psi \Rightarrow \Delta}{\Gamma, \varphi \supset \psi, \Delta} \qquad (\Rightarrow\supset)\ \frac{\Gamma, \varphi \Rightarrow \psi, \Delta}{\Gamma \Rightarrow \varphi \supset \psi, \Delta}$$

$$(\wedge \Rightarrow)\ \frac{\Gamma, \varphi, \psi \Rightarrow \Delta}{\Gamma, \varphi \wedge \psi \Rightarrow \Delta} \quad (\Rightarrow \wedge)\ \frac{\Gamma \Rightarrow \varphi, \Delta \quad \Gamma \Rightarrow \psi, \Delta}{\Gamma \Rightarrow \varphi \wedge \psi, \Delta} \quad (\vee \Rightarrow)\ \frac{\Gamma, \varphi \Rightarrow \Delta \quad \Gamma, \psi \Rightarrow \Delta}{\Gamma, \varphi \vee \psi \Rightarrow \Delta} \quad (\Rightarrow \vee)\ \frac{\Gamma \Rightarrow \varphi, \psi, \Delta}{\Gamma \Rightarrow \varphi \vee \psi, \Delta}$$

Fig. 1. The sequent calculus $\mathsf{G_K}$

Definition 3. A *derivation* of a sequent $\Gamma \Rightarrow \Delta$ in $\mathsf{G_K}$ is a tree in which the nodes are labeled with sequents, the root is labeled with $\Gamma \Rightarrow \Delta$, and every node is the result of an application of some rule of $\mathsf{G_K}$ where the premises are the labels of its children in the tree. A derivation is called an $\mathcal{F}$-*derivation* if it employs only sequents composed of formulas from $\mathcal{F}$. The (K)-*depth* of a derivation is the maximal number of applications of rule (K) in any of the branches of the derivation.

Notation 4. We write $\vdash_{\mathsf{G_K}}^{\mathcal{F},m} \Gamma \Rightarrow \Delta$ if there is a derivation of $\Gamma \Rightarrow \Delta$ in $\mathsf{G_K}$ in which only $\mathcal{F}$-sequents occur and that has (K)-depth at most m. We drop $\mathcal{F}$ from this notation when $\mathcal{F} = \mathcal{L}$; and drop m to dismiss the restriction regarding the (K)-depth.

Example 1. Let $\varphi \overset{\mathsf{def}}{=} \Box(p_1 \wedge p_2) \supset (\Box p_1 \wedge \Box p_2)$ and $\mathcal{F} = sub(\varphi)$. The following is a derivation of $\Rightarrow \varphi$ in $\mathsf{G_K}$ that only uses $\mathcal{F}$-formulas and has (K)-depth of 1 (though the number of applications of (K) in the derivation is 2):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p_1, p_2 \Rightarrow p_1}(\textsc{id})}{p_1 \wedge p_2 \Rightarrow p_1}(\wedge \Rightarrow)}{\Box(p_1 \wedge p_2) \Rightarrow \Box p_1}(\mathrm{K}) \qquad \dfrac{\dfrac{\dfrac{}{p_1, p_2 \Rightarrow p_2}(\textsc{id})}{p_1 \wedge p_2 \Rightarrow p_2}(\wedge \Rightarrow)}{\Box(p_1 \wedge p_2) \Rightarrow \Box p_2}(\mathrm{K})}{\Box(p_1 \wedge p_2) \Rightarrow \Box p_1 \wedge \Box p_2}(\Rightarrow \wedge)}{\Rightarrow \Box(p_1 \wedge p_2) \supset \Box p_1 \wedge \Box p_2}(\Rightarrow\supset)$$

3.2 Semantics

The semantics is based on a four-valued Nmatrix stratified with "levels", where for every m, legal valuations of level $m+1$ are a subset of legal valuations of level m. The underlying Nmatrix, denoted by $\mathsf{M_K}$, is obtained by duplicating the classical truth values. Thus, the sets of truth values and of designated truth values are given by:

$$\mathcal{V}_4 \overset{\mathsf{def}}{=} \{\mathsf{T}, \mathsf{t}, \mathsf{f}, \mathsf{F}\} \qquad\qquad \mathcal{D} \overset{\mathsf{def}}{=} \{\mathsf{T}, \mathsf{t}\}$$

The truth tables are as follows (we have $\overline{\mathcal{D}} = \{\mathsf{f}, \mathsf{F}\}$):

$x \tilde{\supset} y$	T	t	F	f
T	$\mathcal{D}$	$\mathcal{D}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$
t	$\mathcal{D}$	$\mathcal{D}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$
F	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$
f	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$

$x \tilde{\wedge} y$	T	t	F	f
T	$\mathcal{D}$	$\mathcal{D}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$
t	$\mathcal{D}$	$\mathcal{D}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$
F	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$
f	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$

$x \tilde{\vee} y$	T	t	F	f
T	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$
t	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$
F	$\mathcal{D}$	$\mathcal{D}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$
f	$\mathcal{D}$	$\mathcal{D}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}}$

x	$\tilde{\neg} x$
T	$\overline{\mathcal{D}}$
t	$\overline{\mathcal{D}}$
F	$\mathcal{D}$
f	$\mathcal{D}$

x	$\tilde{\Box} x$
T	$\mathcal{D}$
t	$\overline{\mathcal{D}}$
F	$\mathcal{D}$
f	$\overline{\mathcal{D}}$

We employ the following notations for subsets of truth values:

$$\mathsf{TF} \overset{\mathsf{def}}{=} \{\mathsf{T}, \mathsf{F}\} \qquad \mathsf{tf} \overset{\mathsf{def}}{=} \{\mathsf{t}, \mathsf{f}\}$$

For the classical connectives, the truth tables of M_{K} treat t just like T, and f just like F, and are essentially two-valued—the result is either $\mathcal{D}$ or $\overline{\mathcal{D}}$, and it depends solely on whether the inputs are elements of $\mathcal{D}$ or $\overline{\mathcal{D}}$. Thus, for the language without $\Box$, this Nmatrix provides a (non-economic) four-valued semantics for classical logic.

While the output for $\Box$ is also always $\mathcal{D}$ or $\overline{\mathcal{D}}$, it differentiates between T (that results in $\mathcal{D}$) and t (that results in $\overline{\mathcal{D}}$), and similarly between F and f. In fact, this table is captured by the condition: $\tilde{\Box}(x) \in \mathcal{D}$ iff $x \in \mathsf{TF}$.

Example 2. Let $\mathcal{F} = sub(\varphi)$ where φ is the formula from Example 1. The following valuation v is an $\mathcal{F}$-valuation that is M_{K}-legal:

$$v(p_1) = v(p_2) = \mathsf{f} \quad v(p_1 \wedge p_2) = \mathsf{F} \quad v(\Box p_1) = v(\Box p_2) = v(\Box p_1 \wedge \Box p_2) = \mathsf{F}$$

$$v(\Box(p_1 \wedge p_2)) = \mathsf{T} \quad v(\Box(p_1 \wedge p_2) \supset (\Box p_1 \wedge \Box p_2)) = \mathsf{F}$$

To show that it is M_{K}-legal, one needs to verify that $v(\psi) \in \mathsf{pos\text{-}val}(\psi, \mathsf{M}_{\mathsf{K}}, v)$ for each $\psi \in \mathcal{F}$. For example, $v(p_1) = \mathsf{f} \in \mathcal{V}_4 = \mathsf{pos\text{-}val}(p_1, \mathsf{M}_{\mathsf{K}}, v)$. As another example, since $v(p_1) = \mathsf{f}$, we have that $\mathsf{pos\text{-}val}(\Box p_1, \mathsf{M}_{\mathsf{K}}, v) = \tilde{\Box}(\mathsf{f}) = \{\mathsf{F}, \mathsf{f}\}$, and hence $v(\Box p_1) = \mathsf{F} \in \mathsf{pos\text{-}val}(\Box p_1, \mathsf{M}_{\mathsf{K}}, v)$. Notice that v does not satisfy φ.

The truth table for $\Box$ can be understood via "possible worlds" intuition. Our four truth values are intuitively captured as follows, assuming a given formula ψ and a world w:

- T: ψ holds in w and in every world accessible from w;
- t: ψ holds in w but it does not hold in some world accessible from w;
- F: ψ does not hold in w but does hold in every world accessible from w; and
- f: ψ does not hold in w and it does not hold in some world accessible from w.

In the possible worlds semantics, $\Box\psi$ holds in some world w iff ψ holds in every world that is accessible from w, which intuitively explains the table for $\Box$. Note that non-determinism is inherent here. For example, if ψ holds in w and in every world accessible from w (i.e., ψ has value T), we know that $\Box\psi$ holds in w, but we do not know whether $\Box\psi$ holds in every world accessible from w (thus $\Box\psi$ has value T or t).

Now, the Nmatrix $\mathsf{M}_{\mathtt{K}}$ by itself is not adequate for the modal logic $\mathtt{K}$ (as Examples 1 and 2 demonstrate). What is missing is the relation between the choices we make to resolve non-determinism for different formulas. Continuing with the possible worlds intuition, we observe that if a formula φ follows from a set of formulas Σ that hold in all accessible worlds (i.e., φ follows from formulas whose truth value is T or F), then φ itself should hold in all accessible worlds (i.e., φ's truth value should be T or F). Directly encoding this condition requires us to consider a set $\mathbb{V}$ of $\mathsf{M}_{\mathtt{K}}$-legal $\mathcal{F}$-valuations for which the following holds (recall Notation 2 from Sect. 2):

$$\forall v \in \mathbb{V}.\, \forall \varphi \in \mathcal{F}.\, (v^{-1}[\mathsf{TF}] \vdash_{\mathcal{D}}^{\mathbb{V}} \varphi \implies v(\varphi) \in \mathsf{TF}) \qquad (\textit{necessitation})$$

In turn, to obtain completeness we take a maximal set $\mathbb{V}$ that satisfies the *necessitation* condition. While it is possible to define this set of valuations as the greatest fixpoint of *necessitation*, following previous work, we find it convenient to reach this set using "levels":

Definition 5. The set $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m}$ is inductively defined as follows:

- $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}$ is the set of $\mathsf{M}_{\mathtt{K}}$-legal $\mathcal{F}$-valuations.
- $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m+1} \stackrel{\mathsf{def}}{=} \left\{ v \in \mathbb{V}_{\mathtt{K}}^{\mathcal{F},m} \mid \forall \varphi \in \mathcal{F}.\, v^{-1}[\mathsf{TF}] \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m}} \varphi \implies v(\varphi) \in \mathsf{TF} \right\}$

We also define:

$$\mathbb{V}_{\mathtt{K}}^{\mathcal{F}} \stackrel{\mathsf{def}}{=} \bigcap_{m \geq 0} \mathbb{V}_{\mathtt{K}}^{\mathcal{F},m} \qquad \mathbb{V}_{\mathtt{K}}^{m} \stackrel{\mathsf{def}}{=} \mathbb{V}_{\mathtt{K}}^{\mathcal{L},m} \qquad \mathbb{V}_{\mathtt{K}} \stackrel{\mathsf{def}}{=} \bigcap_{m \geq 0} \mathbb{V}_{\mathtt{K}}^{\mathcal{L},m}$$

Similarly to the idea originated by Kearns in [10], valuations are partitioned into *levels*, which are inductively defined. The first level, $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}$, consists solely of the $\mathsf{M}_{\mathtt{K}}$-legal valuations with domain $\mathcal{F}$. For each $m > 0$, the m'th level is defined as a subset of the $(m-1)$'th level, with an additional constraint: a valuation v from level $m-1$ remains in level m, only if every formula $\varphi \in \mathcal{F}$ entailed (at the $m-1$ level) from the set of formulas that were assigned a value from TF by v, is itself assigned a value from TF by v. As we show below, in the "end" of this process, by taking $\bigcap_{m \geq 0} \mathbb{V}_{\mathtt{K}}^{\mathcal{F},m}$, one obtains the greatest set $\mathbb{V}$ satisfying the *necessitation* condition

Remark 1. The *necessitation* condition is similar to the one provided in [8] to the modal logics $\mathtt{KT}$ and $\mathtt{S4}$. In contrast, the condition from [4,10,14] is simpler and does not involve $v^{-1}[\mathsf{TF}]$ at all, but also does not give rise to decision procedures.

Example 3. Following Example 2, while the formula φ is not satisfied by all valuations in $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}$, it is satisfied by all valuations in $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m}$ for every $m > 0$. In particular, the valuation v from Example 2 is not in $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},1}$: we have $p_1 \wedge p_2 \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}} p_1$ and $v(p_1 \wedge p_2) = \mathsf{F}$ (so $p_1 \wedge p_2 \in v^{-1}[\mathsf{TF}]$), but $v(p_1) = \mathsf{f} \notin \mathsf{TF}$.

For each set $\mathcal{F} \subseteq \mathcal{L}$ and $m \geq 0$, we obtain a consequence relation $\vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m}}$ between sets of $\mathcal{F}$-formulas. Disregarding m, we also obtain the relation $\vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F}}}$ (for every $\mathcal{F}$), which we will show to be sound and complete for $\mathtt{K}$. We note that all these relations are compact. The proof of the following theorem relies on the completeness theorems that we prove in Sect. 4.

Theorem 1 (Compactness).

1. *For every $m \geq 0$, if $L \vdash_{\mathcal{D}}^{\mathbb{V}_{K}^{\mathcal{F},m}} R$, then $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{K}^{\mathcal{F},m}} \Delta$ for some finite $\Gamma \subseteq L$ and $\Delta \subseteq R$.*
2. *If $L \vdash_{\mathcal{D}}^{\mathbb{V}_{K}^{\mathcal{F}}} R$, then $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{K}^{\mathcal{F}}} \Delta$ for some finite $\Gamma \subseteq L$ and $\Delta \subseteq R$.*

Now, to show that $\mathbb{V}_{\mathtt{K}}^{\mathcal{F}}$ is indeed the largest set $\mathbb{V}$ of $\mathsf{M}_{\mathtt{K}}$-legal $\mathcal{F}$-valuations that satisfies *necessitation*, we use the following two lemmas. The first is a general construction that relies only on the use of *finite-valued* valuation functions.

Lemma 2. *Let $v_0, v_1, v_2, \dots$ be an infinite sequence of valuations over a common domain $\mathcal{F}$. Then, there exists some v such that for every finite set $\mathcal{F}' \subseteq \mathcal{F}$ of formulas and $m \geq 0$, we have $v|_{\mathcal{F}'} = v_k|_{\mathcal{F}'}$ for some $k \geq m$.*

Proof (Outline). First, if $\mathcal{F}$ is finite, then there is only a finite number of $\mathcal{F}$-valuations, and there must exists some $\mathcal{F}$-valuation v_m that occurs infinitely often in the sequence $v_0, v_1, \dots$. We take $v = v_m$, and the required property trivially holds. Now, assume that $\mathcal{F}$ is infinite, and let $\varphi_0, \varphi_1, \dots$ be an enumeration of the formulas in $\mathcal{F}$. For every $i \geq 0$, let $\mathcal{F}_i = \{\varphi_0, \dots, \varphi_i\}$. We construct a sequence of infinite sets $A_0, A_1, \dots \subseteq \mathbb{N}$ such that:

- For every $i \geq 0$, $A_{i+1} \subseteq A_i$.
- For every $0 \leq j \leq i$, $a \in A_j$, and $b \in A_i$, $v_a(\varphi_j) = v_b(\varphi_j)$.

To do so, take some infinite set $A_0 \subseteq \mathbb{N}$ such that $v_a(\varphi_0) = v_b(\varphi_0)$ for every $a, b \in A_0$ (such set must exist since we have a finite number of truth values). Then, given A_i, we let A_{i+1} be some infinite subset of A_i such that $v_a(\varphi_{i+1}) = v_b(\varphi_{i+1})$ for every $a, b \in A_{i+1}$. The valuation v is defined by $v(\varphi_i) = v_a(\varphi_i)$ for some $a \in A_i$. The properties of the A_i's ensure that v is well defined, and it can be shown that it also satisfies the required property. □

Using Lemma 2 and the compactness property, we can show the following:

Lemma 3. *Let $v_0, v_1, \dots$ be a sequence of valuations over a common domain $\mathcal{F}$ such that $v_m \in \mathbb{V}_{K}^{\mathcal{F},m}$ for every $m \geq 0$. Then, there exists some $v \in \mathbb{V}_{K}^{\mathcal{F}}$ such that for every $\varphi \in \mathcal{F}$, $v(\varphi) = v_m(\varphi)$ for some $m \geq 0$.*

Proof (Outline). By Lemma 2, there exists some v such that for every finite set $\mathcal{F}'$ of formulas, $v|_{\mathcal{F}'} = v_m|_{\mathcal{F}'}$ for some $m \geq 0$. It is easy to verify that v satisfies the required properties. In particular, one shows that $v \in \mathbb{V}_{\mathtt{K}}^{\mathcal{F},m}$ for every $m \geq 0$ by induction on m. In that proof we use Theorem 1 to obtain a finite $\Gamma \subseteq v^{-1}[\mathsf{TF}]$ such that $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m-1}} \varphi$ from the assumption that $v^{-1}[\mathsf{TF}] \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m-1}} \varphi$. Then, the above property of v is applied with $\mathcal{F}' = \Gamma \cup \{\varphi\}$. □

Now, our characterization theorem easily follows:

Theorem 2. *The set $\mathbb{V}_K^{\mathcal{F}}$ is the largest set $\mathbb{V}$ of M_K-legal $\mathcal{F}$-valuations that satisfies necessitation.*

Proof (Outline). To prove that $\mathbb{V}_{\mathtt{K}}^{\mathcal{F}}$ satisfies *necessitation*, one needs to prove that if $v^{-1}[\mathsf{TF}] \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F}}} \varphi$, then also $v^{-1}[\mathsf{TF}] \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m}} \varphi$ for some $m \geq 0$. This is done using Lemma 3. For maximality, given a set $\mathbb{V}$, we assume by contradiction that there is some m such that $\mathbb{V} \not\subseteq \mathbb{V}_{\mathtt{K}}^{\mathcal{F},m}$, take a minimal such m, and show that it cannot be 0. Then, from $\mathbb{V} \subseteq \mathbb{V}_{\mathtt{K}}^{\mathcal{F}} m-1$, it follows that actually $\mathbb{V} \subseteq \mathbb{V}_{\mathtt{K}}^{\mathcal{F},m}$, and thus we obtain a contradiction. □

Finite Domain. By definition we have $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0} \supseteq \mathbb{V}_{\mathtt{K}}^{\mathcal{F},1} \supseteq \mathbb{V}_{\mathtt{K}}^{\mathcal{F},2} \supseteq \ldots$ (and so, $\vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}} \subseteq \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},1}} \subseteq \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},2}} \subseteq \ldots$). Next, we show that when $\mathcal{F}$ is finite, then this sequence must converge.

Lemma 4. *Suppose that $\mathbb{V}_K^{\mathcal{F},m} = \mathbb{V}_K^{\mathcal{F},m+1}$ for some $m \geq 0$. Then, $\mathbb{V}_K^{\mathcal{F}} = \mathbb{V}_K^{\mathcal{F},m}$.*

Lemma 5. *For a* finite *set $\mathcal{F}$ of formulas, $\mathbb{V}_K^{\mathcal{F}} = \mathbb{V}_K^{\mathcal{F},4^{|\mathcal{F}|}}$.*

Proof. The left-to-right inclusion follows from our definitions. For the right-to-left inclusion, note that by Lemma 4, $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m} = \mathbb{V}_{\mathtt{K}}^{\mathcal{F},m+1}$ implies that $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m} = \mathbb{V}_{\mathtt{K}}^{\mathcal{F},k}$ for every $k \geq m$. Thus, it suffices to show that $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},m} = \mathbb{V}_{\mathtt{K}}^{\mathcal{F},m+1}$ for some $0 \leq m \leq 4^{|\mathcal{F}|} + 1$. Indeed, otherwise we have $\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0} \supset \mathbb{V}_{\mathtt{K}}^{\mathcal{F},1} \supset \mathbb{V}_{\mathtt{K}}^{\mathcal{F},2} \supset \ldots \supset \mathbb{V}_{\mathtt{K}}^{\mathcal{F},4^{|\mathcal{F}|}+1}$, but this is impossible since there are only $4^{|\mathcal{F}|}$ functions from $\mathcal{F}$ to $\mathcal{V}_4$. □

Optimized Tables. Starting from level 1, the condition on valuations allows us to refine the truth tables of $\mathsf{M}_{\mathtt{K}}$, and reduce the search space for countermodels. For instance, since $\psi \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}} \varphi \supset \psi$ (for every $\mathcal{F}$ with $\{\psi, \varphi, \varphi \supset \psi\} \subseteq \mathcal{F}$), at level 1 we have that if $\psi \in v^{-1}[\mathsf{TF}]$, then $v(\varphi \supset \psi) \in \mathsf{TF}$. This allows us to remove t and f from the first and third columns (when $y \in \mathsf{TF}$) in the table presenting $\tilde{\supset}$. The following entailments (at level 0), all with a single occurrence of some connective, lead to similar refinements, resulting in the optimized tables below for $\supset$, $\wedge$ and $\vee$:

$$\varphi, \varphi \supset \psi \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}} \psi \qquad \varphi, \psi \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}} \varphi \wedge \psi \qquad \varphi \wedge \psi \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}} \varphi \qquad \varphi \wedge \psi \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}} \psi$$

$$\varphi \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}} \varphi \vee \psi \qquad \psi \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{K}}^{\mathcal{F},0}} \varphi \vee \psi$$

$x \tilde{\supset} y$	T	t	F	f
T	{T}	{t}	{F}	{f}
t	{T}	$\mathcal{D}$	{F}	$\overline{\mathcal{D}}$
F	{T}	{t}	{T}	{t}
f	{T}	$\mathcal{D}$	{T}	$\mathcal{D}$

$x \tilde{\wedge} y$	T	t	F	f
T	{T}	{t}	{F}	{f}
t	{t}	{t}	{f}	{f}
F	{F}	{f}	{F}	{f}
f	{f}	{f}	{f}	{f}

$x \tilde{\vee} y$	T	t	F	f
T	{T}	{T}	{T}	{T}
t	{T}	$\mathcal{D}$	{T}	$\mathcal{D}$
F	{T}	{T}	{F}	{F}
f	{T}	$\mathcal{D}$	{F}	$\overline{\mathcal{D}}$

We note that level 1 valuations are not fully captured by these tables. For example, they must assign T to every formula of the form $\varphi \supset \varphi$, while the table above allows also t when $v(\varphi) \in \mathsf{tf}$. A decision procedure for $\mathtt{K}$ can benefit from relying on these optimized tables instead of the original ones, starting from level 1.

4 Soundness and Completeness

In this section we establish the soundness and completeness of the proposed semantics. For that matter, we first extend the notion of satisfaction to sequents:

Definition 6. An $\mathcal{F}$-valuation v $\mathcal{D}$-*satisfies* an $\mathcal{F}$-sequent $\Gamma \Rightarrow \Delta$, denoted by $v \models_{\mathcal{D}} \Gamma \Rightarrow \Delta$, if $v \not\models_{\mathcal{D}} \varphi$ for some $\varphi \in \Gamma$ or $v \models_{\mathcal{D}} \varphi$ for some $\varphi \in \Delta$.

To prove soundness, we first note that except for (K), the soundness of each derivation rule easily follows from the Nmatrix semantics:

Lemma 6 (Local Soundness). *Consider an application of a rule of* G_K *other than* (K) *deriving a sequent* $\Gamma \Rightarrow \Delta$ *from sequents* $\Gamma_1 \Rightarrow \Delta_1, \ldots, \Gamma_n \Rightarrow \Delta_n$*, such that* $\Gamma \cup \Gamma_1 \cup \ldots \cup \Gamma_n \cup \Delta \cup \Delta_1 \cup \ldots \cup \Delta_n \subseteq \mathcal{F}$*. Let* $v \in \mathbb{V}_K^{\mathcal{F},m}$ *for some* $m \geq 0$*. If* $v \models_{\mathcal{D}} \Gamma_i \Rightarrow \Delta_i$ *for every* $1 \leq i \leq n$*, then* $v \models_{\mathcal{D}} \Gamma \Rightarrow \Delta$*.*

For (K), we make use of the level requirement, and prove the following lemma.

Lemma 7 (Soundness of (K)). *Suppose that* $\Gamma \cup \Box\Gamma \cup \{\varphi, \Box\varphi\} \subseteq \mathcal{F}$*, and* $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_K^{\mathcal{F},m-1}} \varphi$*. Then,* $\Box\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_K^{\mathcal{F},m}} \Box\varphi$*.*

Proof. Let $v \in \mathbb{V}_{\mathsf{K}}^{\mathcal{F},m}$ such that $v \models_{\mathcal{D}} \Box\Gamma$. We prove that $v \models_{\mathcal{D}} \Box\varphi$. By the truth table of $\Box$, we have that $v(\psi) \in \mathsf{TF}$ for every $\psi \in \Gamma$, and we need to show that $v(\varphi) \in \mathsf{TF}$. Since $v(\psi) \in \mathsf{TF}$ for every $\psi \in \Gamma$, we have $\Gamma \subseteq v^{-1}[\mathsf{TF}]$. Since $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathsf{K}}^{\mathcal{F},m-1}} \varphi$, we have $v^{-1}[\mathsf{TF}] \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathsf{K}}^{\mathcal{F},m-1}} \varphi$. Since $v \in \mathbb{V}_{\mathsf{K}}^{\mathcal{F},m}$, it follows that $v(\varphi) \in \mathsf{TF}$. □

The above two lemmas together establish soundness, and from soundness for each level, we easily derive soundness for arbitrary (K)-depth.

Theorem 3 (Soundness for m). *If* $\vdash_{\mathsf{G}_K}^{\mathcal{F},m} \Gamma \Rightarrow \Delta$*, then* $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_K^{\mathcal{F},m}} \Delta$*.*

Theorem 4 (Soundness without m). *If* $\vdash_{\mathsf{G}_K}^{\mathcal{F}} \Gamma \Rightarrow \Delta$*, then* $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_K^{\mathcal{F}}} \Delta$*.*

By taking $\mathcal{F} = \mathcal{L}$ in Theorem 4 we get that if $\vdash_{\mathsf{G}_{\mathsf{K}}} \Gamma \Rightarrow \Delta$, then $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathsf{K}}} \Delta$.

Next, we prove the following two completeness theorems:

Theorem 5 (Completeness for m). *Let* $\mathcal{F} \subseteq \mathcal{L}$ *closed under subformulas and* $\Gamma \Rightarrow \Delta$ *an* $\mathcal{F}$*-sequent. If* $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_K^{\mathcal{F},m}} \Delta$*, then* $\vdash_{\mathsf{G}_K}^{\mathcal{F},m} \Gamma \Rightarrow \Delta$*.*

Theorem 6 (Completeness without m). *Let $\mathcal{F} \subseteq \mathcal{L}$ closed under subformulas and $\Gamma \Rightarrow \Delta$ an $\mathcal{F}$-sequent. If $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{K}^{\mathcal{F}}} \Delta$, then $\vdash_{\mathsf{G}_{K}}^{\mathcal{F}} \Gamma \Rightarrow \Delta$.*

In fact, since $\mathcal{F}$ may be infinite, we need to prove stronger theorems than Theorems 5 and 6, that incorporate infinite sequents.

Definition 7. An ω-sequent is a pair $\langle L, R\rangle$, denoted by $L \Rightarrow R$, such that L and R are (possibly infinite) sets of formulas. We write $\Vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m} L \Rightarrow R$ if $\vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m} \Gamma \Rightarrow \Delta$ for some finite $\Gamma \subseteq L$ and $\Delta \subseteq R$.

Other notions for sequents (e.g., being an $\mathcal{F}$-sequent) are extended to ω-sequents in the obvious way. In particular, $v \models_{\mathcal{D}} L \Rightarrow R$ if $v(\psi) \notin \mathcal{D}$ for some $\psi \in L$ or $v(\psi) \in \mathcal{D}$ for some $\psi \in R$.

Theorem 7 (ω-Completeness for m). *Let $\mathcal{F} \subseteq \mathcal{L}$ closed under subformulas and $L \Rightarrow R$ an ω-$\mathcal{F}$-sequent. If $L \vdash_{\mathcal{D}}^{\mathbb{V}_{K}^{\mathcal{F},m}} R$, then $\Vdash_{\mathsf{G}_{K}}^{\mathcal{F},m} L \Rightarrow R$.*

Theorem 8 (ω-Completeness without m). *Let $\mathcal{F} \subseteq \mathcal{L}$ closed under subformulas and $L \Rightarrow R$ an ω-$\mathcal{F}$-sequent. If $L \vdash_{\mathcal{D}}^{\mathbb{V}_{K}^{\mathcal{F}}} R$, then $\Vdash_{\mathsf{G}_{K}}^{\mathcal{F}} L \Rightarrow R$.*

Theorem 5 is a consequence of Theorem 7. Indeed, by Theorem 7, $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{K}^{\mathcal{F},m}} \Delta$ implies that $\vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m} \Gamma' \Rightarrow \Delta'$ for some (finite) $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$. Using (WEAK), we obtain that $\vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m} \Gamma \Rightarrow \Delta$. Similarly, Theorem 6 is a consequence of Theorem 8. Also, using Lemma 3, we obtain Theorem 8 from Theorem 7. Hence in the remainder of this section we focus on the proof of Theorem 7.

Proof of Theorem 7. We start by defining maximal and consistent ω-sequents, and proving their existence.

Definition 8 (Maximal and consistent ω-sequent). Let $\mathcal{F} \subseteq \mathcal{L}$ and $m \geq 0$. An $\mathcal{F}$-ω-sequent $L \Rightarrow R$ is called:

1. *$\mathcal{F}$-maximal* if $\mathcal{F} \subseteq L \cup R$.
2. *$\langle \mathsf{G}_{\mathsf{K}}, \mathcal{F}, m\rangle$-consistent* if $\nVdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m} L \Rightarrow R$.
3. *$\langle \mathsf{G}_{\mathsf{K}}, \mathcal{F}, m\rangle$-maximal-consistent* (in short, *$\langle \mathsf{G}_{\mathsf{K}}, \mathcal{F}, m\rangle$-max-con*) if it is $\mathcal{F}$-maximal and $\langle \mathsf{G}_{\mathsf{K}}, \mathcal{F}, m\rangle$-consistent.

Lemma 8. *Let $\mathcal{F} \subseteq \mathcal{L}$ and $L \Rightarrow R$ an $\mathcal{F}$-ω-sequent. Suppose that $\nVdash_{\mathsf{G}_{K}}^{\mathcal{F},m} L \Rightarrow R$. Then, there exist sets $L_{MC(\mathsf{G}_{K},\mathcal{F},m,L\Rightarrow R)}$ and $R_{MC(\mathsf{G}_{K},\mathcal{F},m,L\Rightarrow R)}$ such that the following hold:*

- *$L \subseteq L_{MC(\mathsf{G}_{K},\mathcal{F},m,L\Rightarrow R)}$ and $R \subseteq R_{MC(\mathsf{G}_{K},\mathcal{F},m,L\Rightarrow R)}$.*
- *$L_{MC(\mathsf{G}_{K},\mathcal{F},m,L\Rightarrow R)} \cup R_{MC(\mathsf{G}_{K},\mathcal{F},m,L\Rightarrow R)} \subseteq \mathcal{F}$.*
- *$L_{MC(\mathsf{G}_{K},\mathcal{F},m,L\Rightarrow R)} \Rightarrow R_{MC(\mathsf{G}_{K},\mathcal{F},m,L\Rightarrow R)}$ is $\langle \mathsf{G}_{K}, \mathcal{F}, m\rangle$-max-con.*

Thus, given an underivable ω-sequent, we can extend it to a $\langle \mathsf{G}_{\mathsf{K}}, \mathcal{F}, m\rangle$-max-con ω-sequent. This ω-sequent induces the canonical countermodel, as defined next.

Algorithm 1. Deciding $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathsf{K}}} \varphi$.

```
$\mathcal{F} \leftarrow sub(\Gamma \cup \{\varphi\})$
$m \leftarrow 4^{|\mathcal{F}|}$
for $v \in \mathbb{V}_{\mathsf{K}}^{\mathcal{F},m}$ do
    if $v \models_{\mathcal{D}} \Gamma$ and $v \not\models_{\mathcal{D}} \varphi$ then
        return (“NO”, $v$)
return “YES”
```

Notation 9. We denote the set $\{\psi \in \mathcal{F} \mid \Box\psi \in X\}$ by $\mathbb{B}_{\mathcal{F}}^{X}$.

Definition 10. Suppose that $L \uplus R = \mathcal{F}$. The *canonical model w.r.t.* $L \Rightarrow R$*,* $\mathcal{F}$*, and* m, denoted by $v(\mathcal{F}, L \Rightarrow R, m)$, is the $\mathcal{F}$-valuation defined as follows (in λ notation):

For $m = 0$:

$$\lambda\varphi \in \mathcal{F}. \begin{cases} \mathsf{T} & \varphi \in L \text{ and } \Box\varphi \in L \\ \mathsf{t} & \varphi \in L \text{ and } \Box\varphi \notin L \\ \mathsf{F} & \varphi \in R \text{ and } \Box\varphi \in L \\ \mathsf{f} & \varphi \in R \text{ and } \Box\varphi \notin L \end{cases}$$

For $m > 0$:

$$\lambda\varphi \in \mathcal{F}. \begin{cases} \mathsf{T} & \varphi \in L \text{ and } \Vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \mathbb{B}_{\mathcal{F}}^{L} \Rightarrow \varphi \\ \mathsf{t} & \varphi \in L \text{ and } \nVdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \mathbb{B}_{\mathcal{F}}^{L} \Rightarrow \varphi \\ \mathsf{F} & \varphi \in R \text{ and } \Vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \mathbb{B}_{\mathcal{F}}^{L} \Rightarrow \varphi \\ \mathsf{f} & \varphi \in R \text{ and } \nVdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \mathbb{B}_{\mathcal{F}}^{L} \Rightarrow \varphi \end{cases}$$

Clearly, $v(\mathcal{F}, L \Rightarrow R, m) \not\models_{\mathcal{D}} L \Rightarrow R$. The proof of Theorem 7 is done by induction on m, and then carries on by showing that if $L \Rightarrow R$ is $\langle \mathsf{G}_{\mathsf{K}}, \mathcal{F}, m\rangle$-max-con, then $v(\mathcal{F}, L \Rightarrow R, m)$ belongs to $\mathbb{V}_{\mathsf{K}}^{\mathcal{F},m}$ for every m.

Concretely, let $v \overset{\text{def}}{=} v(\mathcal{F}, L \Rightarrow R, m)$. We show that $v \in \mathbb{V}_{\mathsf{K}}^{\mathcal{F},k}$ for every $k \leq m$ by induction on k. The base case $k = 0$ is straightforward. For $k > 0$, we have $v \in \mathbb{V}_{\mathsf{K}}^{\mathcal{F},k-1}$ by the induction hypothesis. Let $\varphi \in \mathcal{F}$, and suppose that $v^{-1}[\mathsf{TF}] \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathsf{K}}^{\mathcal{F},k-1}} \varphi$. To show that $v(\varphi) \in \mathsf{TF}$, we prove that $\Vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \mathbb{B}_{\mathcal{F}}^{L} \Rightarrow \varphi$. By the outer induction hypothesis (regarding the completeness theorem itself), $v^{-1}[\mathsf{TF}] \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathsf{K}}^{\mathcal{F},k-1}} \varphi$ implies that $\Vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},k-1} v^{-1}[\mathsf{TF}] \Rightarrow \varphi$, which implies that $\Vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} v^{-1}[\mathsf{TF}] \Rightarrow \varphi$. Hence, there is a finite set $\{\varphi_1, \ldots, \varphi_n\} \subseteq v^{-1}[\mathsf{TF}]$ such that $\vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \{\varphi_1, \ldots, \varphi_n\} \Rightarrow \varphi$. For every $1 \leq i \leq n$, since $\varphi_i \in v^{-1}[\mathsf{TF}]$, we have that $\Vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \mathbb{B}_{\mathcal{F}}^{L} \Rightarrow \varphi_i$ and hence $\vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \Gamma_i \Rightarrow \varphi_i$ for some $\Gamma_i \subseteq \mathbb{B}_{\mathcal{F}}^{L}$. Using n applications of (CUT) on these sequents and $\vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \{\varphi_1, \ldots, \varphi_n\} \Rightarrow \varphi$, we obtain that $\vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \Gamma_1, \ldots, \Gamma_n \Rightarrow \varphi$, and so $\Vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m-1} \mathbb{B}_{\mathcal{F}}^{L} \Rightarrow \varphi$.

5 Effectiveness of the Semantics

In this section we study the effectiveness of the semantics introduced in Definition 5 for deciding $\vdash_{\mathsf{M}_{\mathsf{K}}}$. Roughly speaking, a semantic framework is said to be *effective* if it induces a decision procedure that decides its underlying logic.

Consider Algorithm 1. Given a finite set Γ of formulas and a formula φ, it checks whether any valuations in $\mathbb{V}_{\mathsf{K}}^{\mathcal{F},m}$ is a countermodel. The correctness of this algorithm relies on the analyticity of G_{K}, namely:

Lemma 9 ([11]). *If* $\vdash_{\mathsf{G}_K} \Gamma \Rightarrow \Delta$*, then* $\vdash^{sub(\Gamma\cup\{\varphi\})}_{\mathsf{G}_K} \Gamma \Rightarrow \Delta$*.*

Using Lemma 9, we show that the algorithm is correct.

Lemma 10. *Algorithm 1 always terminates, and returns "YES" iff* $\Gamma \vdash^{\mathbb{V}_K}_{\mathcal{D}} \varphi$*.*

Proof. Termination follows from the fact that $\mathbb{V}^{\mathcal{F},m}_{\mathsf{K}}$ is finite. Suppose that the result is "YES" and assume for contradiction that $\Gamma \nvdash^{\mathbb{V}_{\mathsf{K}}}_{\mathcal{D}} \varphi$. Hence, there exists some $u \in \mathbb{V}_{\mathsf{K}}$ such that $u \models_{\mathcal{D}} \Gamma$ and $u \not\models_{\mathcal{D}} \varphi$. Consider $v \overset{\mathsf{def}}{=} u|_{\mathcal{F}}$. Then, $v \in \mathbb{V}^{\mathcal{F}}_{\mathsf{K}} \subseteq \mathbb{V}^{\mathcal{F},m}_{\mathsf{K}}$, which contradicts the fact that the algorithm returns "YES". Now, suppose that the result is "NO". Then, there exists some $v \in \mathbb{V}^{\mathcal{F},m}_{\mathsf{K}}$ such that $v \models_{\mathcal{D}} \Gamma$ and $v \not\models_{\mathcal{D}} \varphi$. By Lemma 5, $v \in \mathbb{V}^{\mathcal{F}}_{\mathsf{K}}$. Hence, $\Gamma \nvdash^{\mathbb{V}^{\mathcal{F}}_{\mathsf{K}}}_{\mathcal{D}} \varphi$. By Theorem 3, we have $\nvdash^{\mathcal{F}}_{\mathsf{G}_{\mathsf{K}}} \Gamma \Rightarrow \varphi$. By Lemma 9, we have $\nvdash_{\mathsf{G}_{\mathsf{K}}} \Gamma \Rightarrow \varphi$. By Theorem 6, we have $\Gamma \nvdash^{\mathbb{V}_{\mathsf{K}}}_{\mathcal{D}} \varphi$. □

Lemma 10 shows that Algorithm 1 is a decision procedure for $\vdash_{\mathsf{M}_{\mathsf{K}}}$, when ignoring the additional output provided in Line 5. However, it is typical in applications that a "YES" or "NO" answer is not enough, and often it is expected that a "NO" result is accompanied with a countermodel. Algorithm 1 returns a valuation v in case the answer is "NO", but Lemma 10 does not ensure that v is indeed a countermodel for $\Gamma \vdash^{\mathbb{V}_{\mathsf{K}}}_{\mathcal{D}} \varphi$. The issue is that the valuation v from the proof of Lemma 10 witnesses the fact that $\nvdash^{\mathbb{V}_{\mathsf{K}}}_{\mathcal{D}}$ only in a non-constructive way. Indeed, using the soundness and completeness theorems, we are able to deduce that $v' \models_{\mathcal{D}} \Gamma$ and $v' \not\models_{\mathcal{D}} \varphi$ for some $v' \in \mathbb{V}_{\mathsf{K}}$, but the relation between v and v' is unclear. Most importantly, it is not clear whether v and v' agree on $\mathcal{F}$-formulas. In the remainder of this section we prove that v' extends v, and so the returned countermodel of Line 5 can be trusted.

We say that a valuation v' *extends* a valuation v if $\mathsf{Dom}(v) \subseteq \mathsf{Dom}(v')$ and $v'(\varphi) = v(\varphi)$ for every $\varphi \in \mathsf{Dom}(v)$ (identifying functions with sets of pairs, this means $v \subseteq v'$). Clearly, for a $\mathsf{Dom}(v)$-formula ψ we have that $v' \models_{\mathcal{D}} \psi$ iff $v \models_{\mathcal{D}} \psi$. We first show how to extend a given valuation $v \in \mathbb{V}^{\mathcal{F},m}_{\mathsf{K}}$ by a single formula ψ such that $sub(\psi)\setminus\{\psi\} \subseteq \mathcal{F}$, obtaining a valuation $v' \in \mathbb{V}^{\mathcal{F}\cup\{\psi\},m}_{\mathsf{K}}$ that agrees with v on all formulas in $\mathcal{F}$.

Lemma 11. *Let* $m \geq 0$*,* $\mathcal{F} \subseteq \mathcal{L}$*, and* $v \in \mathbb{V}^{\mathcal{F},m}_{K}$*. Let* $\psi \in \mathcal{L}\setminus\mathcal{F}$ *such that* $sub(\psi)\setminus\{\psi\} \subseteq \mathcal{F}$*. Then,* v *can be extended to some* $v' \in \mathbb{V}^{\mathcal{F}\cup\{\psi\},m}_{K}$*.*

We sketch the proof of Lemma 11.

When $m = 0$, v' exists from Lemma 1. For $m > 0$, we define v' as follows:[1]

$$v' \overset{\mathsf{def}}{=} \lambda\varphi \in \mathcal{F}\cup\{\psi\}\,.\begin{cases} v(\varphi) & \varphi \in \mathcal{F} \\ \min(\mathsf{pos\text{-}val}(\psi, \mathsf{M}_{\mathsf{K}}, v) \cap \mathsf{TF}) & \varphi = \psi \wedge v^{-1}[\mathsf{TF}] \vdash^{\mathbb{V}^{\mathcal{F}\cup\{\psi\},m-1}_{\mathsf{K}}}_{\mathcal{D}} \psi \\ \min(\mathsf{pos\text{-}val}(\psi, \mathsf{M}_{\mathsf{K}}, v) \cap \mathsf{tf}) & \text{otherwise} \end{cases}$$

[1] The use of min here assumes an arbitrary order on truth values. It is used here only to choose *some* element from a non-empty set of truth values.

The proof of Lemma 11 then carries on by showing that $v' \in \mathbb{V}_{\mathsf{K}}^{\mathcal{F}\cup\{\psi\},m}$.

Next, Lemma 11 is used in order to extend partial valuations into total ones.

Lemma 12. *Let $v \in \mathbb{V}_{K}^{\mathcal{F},m}$ for some $\mathcal{F}$ closed under subformulas. Then, v can be extended to some $v' \in \mathbb{V}_{K}^{m}$.*

Finally, Lemmas 3 and 12 can be used in order to extend any partial valuation in $\mathbb{V}_{\mathsf{K}}^{\mathcal{F}}$ into a total one.

Lemma 13. *Let $v \in \mathbb{V}_{K}^{\mathcal{F}}$ for some set $\mathcal{F}$ closed under subformulas. Then, v can be extended to some $v' \in \mathbb{V}_{K}$.*

We conclude by showing that when Algorithm 1 returns (“NO”, v), then v is a finite representation of a true countermodel for $\Gamma \vdash_{\mathsf{M}_{\mathsf{K}}} \varphi$.

Corollary 1. *If $\Gamma \not\vdash_{\mathcal{D}}^{\mathbb{V}_{K}} \varphi$. Then Algorithm 1 returns (“NO”, v) for some v for which there exists $v' \in \mathbb{V}_{K}$ such that $v = v'|_{sub(\Gamma\cup\{\varphi\})}$, $v' \models_{\mathcal{D}} \Gamma$, and $v' \not\models_{\mathcal{D}} \varphi$.*

Proof. Suppose that $\Gamma \not\vdash_{\mathcal{D}}^{\mathbb{V}_{\mathsf{K}}} \varphi$. Then by Lemma 10, Algorithm 1 does not return “YES”. Therefore, it returns (“NO”, v) for some $v \in \mathbb{V}_{\mathsf{K}}^{\mathcal{F},m}$ such that $v \models_{\mathcal{D}} \Gamma$ and $v \not\models_{\mathcal{D}} \varphi$, where $\mathcal{F} = sub(\Gamma \cup \{\varphi\})$ and $m = 4^{|\mathcal{F}|}$. By Lemma 5, $v \in \mathbb{V}_{\mathsf{K}}\mathcal{F}$. By Lemma 13, v can be extended to some $v' \in \mathbb{V}_{\mathsf{K}}$. Therefore, $v = v'|_{sub(\Gamma\cup\{\varphi\})}$, $v' \models_{\mathcal{D}} \Gamma$, and $v' \not\models_{\mathcal{D}} \varphi$. □

Remark 2. Notice that in scenarios where model generation is not important, m can be set to a much smaller number in Line 2 of Algorithm 1, namely, the “modal depth” of the input.[2] The reason for that is that for such m, it can be shown that $\vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F},m} \Gamma \Rightarrow \varphi$ iff $\vdash_{\mathsf{G}_{\mathsf{K}}}^{\mathcal{F}} \Gamma \Rightarrow \varphi$, by reasoning about the applications of rule (K). Using the soundness and completeness theorems, we can get $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathsf{K}}^{\mathcal{F},m}} \varphi$ iff $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathsf{K}}^{\mathcal{F}}} \varphi$, and so limiting to such m is enough. Notice however, that we do not necessarily get $\mathbb{V}_{\mathsf{K}}^{\mathcal{F},m} = \mathbb{V}_{\mathsf{K}}^{\mathcal{F}}$ for such m, and so the valuation returned in Line 5 might not be an element of $\mathbb{V}_{\mathsf{K}}^{\mathcal{F}}$.

6 The Modal Logic KT

In this section we obtain similar results for the modal logic KT. First, the calculus G_{KT} is obtained from G_{K} by adding the following rule (see, e.g., [16]):

$$(T)\ \frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma, \Box\varphi \Rightarrow \Delta}$$

Derivations are defined as before. (In particular, the (K)-depth of a derivation still depends on applications of rule (K), not of rule (T).) We write $\vdash_{\mathsf{G}_{\mathsf{KT}}}^{\mathcal{F},m} \Gamma \Rightarrow \Delta$

[2] The *modal depth* of an atomic formula p is 0. The modal depth of $\Box\varphi$ is the modal depth of φ plus 1. The modal depth of $\diamond(\varphi_1, \ldots, \varphi_n)$ for $\diamond \neq \Box$ is the maximum among the modal depths of $\varphi_1, \ldots, \varphi_n$.

if there is a derivation of $\Gamma \Rightarrow \Delta$ in $\mathsf{G}_{\mathtt{KT}}$ in which only $\mathcal{F}$-sequents occur and that has (K)-depth at most m.

Next, we consider the semantics. For a valuation $v \in \mathbb{V}_{\mathtt{K}}$ to respect rule (T), we must have that if $v \models_{\mathcal{D}} \Gamma, \varphi \Rightarrow \Delta$, then $v \models_{\mathcal{D}} \Gamma, \Box\varphi \Rightarrow \Delta$. In particular, when $v \not\models_{\mathcal{D}} \Gamma \Rightarrow \Delta$, we get that if $v(\varphi) \notin \mathcal{D}$, then $v(\Box\varphi) \notin \mathcal{D}$. Now, if $v(\varphi) = \mathsf{F}$, then $v(\Box\varphi) \in \mathcal{D}$ according to the truth table of $\Box$ in $\mathsf{M}_{\mathtt{K}}$. But, we must have $v(\Box\varphi) \notin \mathcal{D}$. This leads us to remove F from $\mathsf{M}_{\mathtt{K}}$.

We thus obtain the following Nmatrix $\mathsf{M}_{\mathtt{KT}}$: The sets of truth values and of designated truth values are given by[3]

$$\mathcal{V}_3 \overset{\text{def}}{=} \{\mathsf{T}, \mathsf{t}, \mathsf{f}\} \qquad\qquad \mathcal{D} \overset{\text{def}}{=} \{\mathsf{T}, \mathsf{t}\}$$

and the truth tables are as follows:

$x \tilde{\supset} y$	T	t	f
T	$\mathcal{D}$	$\mathcal{D}$	$\{\mathsf{f}\}$
t	$\mathcal{D}$	$\mathcal{D}$	$\{\mathsf{f}\}$
f	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$

$x \tilde{\wedge} y$	T	t	f
T	$\mathcal{D}$	$\mathcal{D}$	$\{\mathsf{f}\}$
t	$\mathcal{D}$	$\mathcal{D}$	$\{\mathsf{f}\}$
f	$\{\mathsf{f}\}$	$\{\mathsf{f}\}$	$\{\mathsf{f}\}$

$x \tilde{\vee} y$	T	t	f
T	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$
t	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$
f	$\mathcal{D}$	$\mathcal{D}$	$\{\mathsf{f}\}$

x	$\tilde{\neg} x$
T	$\{\mathsf{f}\}$
t	$\{\mathsf{f}\}$
f	$\mathcal{D}$

x	$\tilde{\Box} x$
T	$\mathcal{D}$
t	$\{\mathsf{f}\}$
f	$\{\mathsf{f}\}$

Again, one may gain intuition from the possible worlds semantics. There, the logic $\mathtt{KT}$ is characterized by frames with *reflexive* accessibility relation. Thus, for instance, if ψ holds in w but not in some world accessible from w (i.e., ψ has value t), we know that $\Box\psi$ does not hold in w, and the reflexivity of the accessibility relation implies that $\Box\psi$ does not hold in some world accessible from w (thus $\Box\psi$ has value f).

Example 4. Let $\varphi \overset{\text{def}}{=} \Box\Box(p_1 \wedge p_2) \supset \Box p_1$ and $\mathcal{F} \overset{\text{def}}{=} sub(\varphi)$. The sequent $\Rightarrow \varphi$ has a derivation in $\mathsf{G}_{\mathtt{KT}}$ using only $\mathcal{F}$ formulas of (K)-depth of 1. However, it is not satisfied by all $\mathsf{M}_{\mathtt{KT}}$-legal $\mathcal{F}$-valuations. For example, the following valuation is an $\mathsf{M}_{\mathtt{KT}}$-legal valuation that does not satisfy φ:

$$v(p_1) = v(p_2) = \mathsf{t} \quad v(\Box p_1) = \mathsf{f}$$

$$v(p_1 \wedge p_2) = v(\Box(p_1 \wedge p_2)) = v(\Box\Box(p_1 \wedge p_2)) = \mathsf{T} \quad v(\varphi) = \mathsf{f}$$

Next, we define the levels of valuations for $\mathsf{M}_{\mathtt{KT}}$. These are obtained from Definition 5 by removing the value F:

Definition 11. The set $\mathbb{V}_{\mathtt{KT}}^{\mathcal{F},m}$ is recursively defined as follows:

- $\mathbb{V}_{\mathtt{KT}}^{\mathcal{F},0}$ is the set of $\mathsf{M}_{\mathtt{KT}}$-legal $\mathcal{F}$-valuations.
- $\mathbb{V}_{\mathtt{KT}}^{\mathcal{F},m+1} \overset{\text{def}}{=} \left\{ v \in \mathbb{V}_{\mathtt{KT}}^{\mathcal{F},m} \mid \forall \varphi \in \mathcal{F}.\, v^{-1}[\mathsf{T}] \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathtt{KT}}^{\mathcal{F},m}} \varphi \implies v(\varphi) = \mathsf{T} \right\}$

We also define:

$$\mathbb{V}_{\mathtt{KT}}^{\mathcal{F}} \overset{\text{def}}{=} \bigcap_{m \geq 0} \mathbb{V}_{\mathtt{KT}}^{\mathcal{F},m} \qquad\qquad \mathbb{V}_{\mathtt{KT}}^{m} \overset{\text{def}}{=} \mathbb{V}_{\mathtt{KT}}^{\mathcal{L},m} \qquad\qquad \mathbb{V}_{\mathtt{KT}} \overset{\text{def}}{=} \bigcap_{m \geq 0} \mathbb{V}_{\mathtt{KT}}^{\mathcal{L},m}$$

[3] In this section we denote the set $\{\mathsf{T}\}$ by TF.

Example 5. Following Example 4, we note that for every $v \in \mathbb{V}_{\mathsf{KT}}^{\mathcal{F},m}$ with $m > 0$, we have $v \models_{\mathcal{D}} \varphi$. In particular, the valuation v from Example 4 does not belong to $\mathbb{V}_{\mathsf{KT}}^{\mathcal{F},m}$: $\Box(p_1 \wedge p_2) \in v^{-1}[\mathsf{T}]$, $\Box(p_1 \wedge p_2) \vdash_{\mathcal{D}}^{\mathbb{V}_{\mathsf{KT}}^{\mathcal{F},0}} p_1$, but $v(p_1) = \mathsf{t}$.

Similarly to Theorem 2, the levels of valuations converge to a maximal set that satisfies the following condition:

$$\forall v \in \mathbb{V}. \forall \varphi \in \mathcal{F}. v^{-1}[\mathsf{T}] \vdash_{\mathcal{D}}^{\mathbb{V}} \varphi \implies v(\varphi) = \mathsf{T} \qquad (\mathit{necessitation}_{\mathsf{KT}})$$

Theorem 9. *The set $\mathbb{V}_{KT}^{\mathcal{F}}$ is the largest set $\mathbb{V}$ of M_{KT}-legal $\mathcal{F}$-valuations that satisfies necessitation$_{KT}$.*

The proof of Theorem 9 is analogous to that of Theorem 2.

Remark 3. The *necessitation*$_{\mathsf{KT}}$ condition is equivalent to the one given in [8], except that the underlying truth table is different. Theorem 9 proves that our gradual way of defining $\mathbb{V}_{\mathsf{KT}}^{\mathcal{F}}$ via levels coincides with the semantic condition from [8].

As we demonstrated for K, starting from level 1, the condition on valuations allows us to refine the truth tables of M_{KT}, and reduce the search space. Simple entailments (at level 0) lead to the optimized tables below for $\supset$, $\wedge$ and $\vee$:

$x \tilde{\supset} y$	T	t	f
T	$\{\mathsf{T}\}$	$\{\mathsf{t}\}$	$\{\mathsf{f}\}$
t	$\{\mathsf{T}\}$	$\mathcal{D}$	$\{\mathsf{f}\}$
f	$\{\mathsf{T}\}$	$\mathcal{D}$	$\mathcal{D}$

$x \tilde{\wedge} y$	T	t	f
T	$\{\mathsf{T}\}$	$\{\mathsf{t}\}$	$\{\mathsf{f}\}$
t	$\{\mathsf{t}\}$	$\{\mathsf{t}\}$	$\{\mathsf{f}\}$
f	$\{\mathsf{f}\}$	$\{\mathsf{f}\}$	$\{\mathsf{f}\}$

$x \tilde{\vee} y$	T	t	f
T	$\{\mathsf{T}\}$	$\{\mathsf{T}\}$	$\{\mathsf{T}\}$
t	$\{\mathsf{T}\}$	$\mathcal{D}$	$\mathcal{D}$
f	$\{\mathsf{T}\}$	$\mathcal{D}$	$\{\mathsf{f}\}$

Soundness and completeness for G_{KT} are obtained analogously to G_{K}, keeping in mind that M_{KT} is obtained from M_{K} by deleting the value F. For soundness, this is captured by the rule (T). For completeness, the same construction of a countermodel is performed , while rule (T) ensures that it is three-valued.

Theorem 10 (Soundness and Completeness). *Let $\mathcal{F} \subseteq \mathcal{L}$ closed under subformulas and $\Gamma \Rightarrow \Delta$ an $\mathcal{F}$-sequent.*

1. *For every $m \geq 0$, $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{KT}^{\mathcal{F},m}} \Delta$ iff $\vdash_{\mathsf{G}_{KT}}^{\mathcal{F},m} \Gamma \Rightarrow \Delta$.*
2. *$\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{KT}^{\mathcal{F}}} \Delta$ iff $\vdash_{\mathsf{G}_{KT}}^{\mathcal{F}} \Gamma \Rightarrow \Delta$.*

Effectiveness is also shown similarly to K. For that matter, we use the following main lemma, whose proof is similar to Lemma 13. The only component that is added to that proof is making sure that the constructed model is three-valued.

Lemma 14. *Let $v \in \mathbb{V}_{KT}^{\mathcal{F}}$ for some set $\mathcal{F}$ closed under subformulas. Then, v can be extended to some $v' \in \mathbb{V}_{KT}$.*

Let Algorithm 2 be obtained from Algorithm 1 by setting m to $3^{|\mathcal{F}|}$ in Line 2, and taking $v \in \mathbb{V}_{\mathsf{KT}}^{\mathcal{F},m}$ in Line 3. Similarly to Lemma 10 and Corollary 1, we get that Algorithm 2 is a model-producing decision procedure for $\vdash_{\mathsf{M}_{\mathsf{KT}}}$.

Lemma 15. *Algorithm 2 always terminates, and returns "YES" iff* $\Gamma \vdash_{\mathcal{D}}^{\mathbb{V}_{KT}} \varphi$*. Further, if* $\Gamma \not\vdash_{\mathcal{D}}^{\mathbb{V}_{KT}} \varphi$*, then it returns ("NO",* v*) for some* v *for which there exists* $v' \in \mathbb{V}_{KT}$ *such that* $v = v'|_{sub(\Gamma \cup \{\varphi\})}$*,* $v' \models_{\mathcal{D}} \Gamma$*, and* $v' \not\models_{\mathcal{D}} \varphi$*.*

7 Future Work

We have introduced a new semantics for the modal logic K, based on levels of valuations in many-valued non-deterministic matrices. Our semantics is effective, and was shown to tightly correspond to derivations in a sequent calculus for K. We also adapted these results for the modal logic KT.

There are two main directions for future work. The first is to establish similar semantics for other normal modal logics, such as KD, K4, S4 and S5, and to investigate $\Diamond$ as an independent modality. The second is to analyze the complexity, implement and experiment with decision procedures for K and KT based on the proposed semantics. In particular, we plan to consider SAT-based decision procedures that would encode this semantics in SAT, directly or iteratively.

References

1. Avron, A., Zamansky, A.: Non-deterministic semantics for logical systems - a survey. In: Gabbay, D., Guenther, F. (eds.) Handbook of Philosophical Logic, vol. 16, pp. 227–304. Springer, Dordrecht (2011). https://doi.org/10.1007/978-94-007-0479-4_4
2. Avron, A., Lev, I.: Non-deterministic multi-valued structures. J. Log. Comput. **15**, 241–261 (2005). Conference version: Avron, A., Lev, I.: Canonical propositional Gentzen-type systems. In: International Joint Conference on Automated Reasoning, IJCAR 2001. Proceedings, LNAI, vol. 2083, pp. 529–544. Springer (2001)
3. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, 2nd edn., vol. 336. IOS Press (2021)
4. Coniglio, M.E., del Cerro, L.F., Peron, N.M.: Finite non-deterministic semantics for some modal systems. J. Appl. Non Class. Log. **25**(1), 20–45 (2015)
5. Coniglio, M.E., del Cerro, L.F., Peron, N.M.: Errata and addenda to 'finite non-deterministic semantics for some modal systems'. J. Appl. Non Class. Log. **26**(4), 336–345 (2016)
6. Coniglio, M.E., Toledo, G.V.: Two decision procedures for da costa's Cn logics based on restricted Nmatrix semantics. Stud. Log. **110**, 1–42 (2021)
7. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.: Reasoning About Knowledge. MIT Press, Cambridge (2004)
8. Grätz, L.: Truth tables for modal logics T and S4, by using three-valued non-deterministic level semantics. J. Log. Comput. **32**(1), 129–157 (2022)
9. Halpern, J., Manna, Z., Moszkowski, B.: A hardware semantics based on temporal intervals. In: Diaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 278–291. Springer, Heidelberg (1983). https://doi.org/10.1007/BFb0036915
10. Kearns, J.T.: Modal semantics without possible worlds. J. Symb. Log. **46**(1), 77–86 (1981)

11. Lahav, O., Avron, A.: A unified semantic framework for fully structural propositional sequent systems. ACM Trans. Comput. Log. **14**(4), 271–273 (2013)
12. Pawlowski, P., La Rosa, E.: Modular non-deterministic semantics for T, TB, S4, S5 and more. J. Log. Comput. **32**(1), 158–171 (2022)
13. Pratt, V.R.: Application of modal logic to programming. Stud. Log.: Int. J. Symb. Log. **39**(2/3), 257–274 (1980)
14. Skurt, D., Omori, H.: More modal semantics without possible worlds. FLAP **3**(5), 815–846 (2016)
15. Urquhart, A.: Many-valued logic. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. II, 2nd edn., pp. 249–295. Kluwer (2001)
16. Wansing, H.: Sequent systems for modal logics. In: Gabbay, D.M., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. 8, 2nd edn., pp. 61–145. Springer, Dordrecht (2002). https://doi.org/10.1007/978-94-010-0387-2_2

Local Reductions for the Modal Cube

Cláudia Nalon[1], Ullrich Hustadt[2](✉), Fabio Papacchini[3], and Clare Dixon[4]

[1] Department of Computer Science, University of Brasília, Brasília, Brazil
nalon@unb.br

[2] Department of Computer Science, University of Liverpool, Liverpool, UK
U.Hustadt@liverpool.ac.uk

[3] School of Computing and Communications, Lancaster University in Leipzig, Leipzig, Germany
f.papacchini@lancaster.ac.uk

[4] Department of Computer Science, University of Manchester, Manchester, UK
clare.dixon@manchester.ac.uk

Abstract. The modal logic K is commonly used to represent and reason about necessity and possibility and its extensions with combinations of additional axioms are used to represent knowledge, belief, desires and intentions. Here we present local reductions of all propositional modal logics in the so-called modal cube, that is, extensions of K with arbitrary combinations of the axioms B, D, T, 4 and 5 to a normal form comprising a formula and the set of modal levels it occurs at. Using these reductions we can carry out reasoning for all these logics with the theorem prover K$_S$P. We define benchmarks for these logics and experiment with the reduction approach as compared to an existing resolution calculus with specialised inference rules for the various logics.

1 Introduction

Modal logics have been used to represent and reason about mental attitudes such as knowledge, belief, desire and intention, see for example [17,20,31]. These can be represented using extensions of the basic modal logic K with one or more of the axioms B (symmetry), D (seriality), T (reflexivity), 4 (transitivity) and 5 (Euclideaness). The logic K and these extensions form the so-called *modal cube*, see Fig. 1. In the diagram, a line from a logic L_1 to a logic L_2 to its right and/or above means that all theorems of L_1 are also theorems of L_2, but not vice versa. As indicated in Fig. 1, some of the logics have the same theorems, e.g., KB5 and KB4. Also, all logics not explicitly listed have the same theorems as KT5 aka S5. In total there are 15 distinct logics.

While these modal logics are well-studied and a multitude of calculi and translations to other logics exist, see, e.g., [1,3–6,9,13,14,16,18,22,41], fully

C. Dixon was partially supported by the EPSRC funded RAI Hubs FAIR-SPACE (EP/R026092/1) and RAIN (EP/R026084/1), and the EPSRC funded programme Grant S4 (EP/N007565/1).

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 486–505, 2022.
https://doi.org/10.1007/978-3-031-10769-6_29

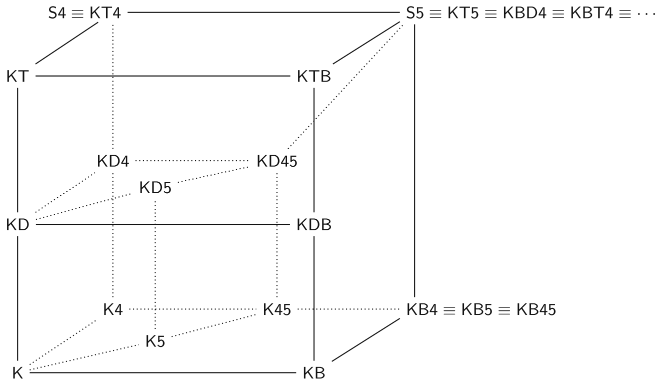


Fig. 1. Modal Cube: Relationships between modal logics

automatic support by provers is still lacking. Early implementations covering the full modal cube, such as Catach's TABLEAUX system [7], are no longer available. LoTREC 2.0 [10] supports a wide range of logics but is not intended as an automatic theorem prover. MOIN [11] supports all the logics but the focus is on producing human-readable proofs and countermodels for small formulae. Other provers that go beyond just K, like MleanCoP [28] and CEGARBox [15] only support a small subset of the 15 logics. There are also a range of translations from modal logics to first-order and higher-order logics [13,18,19,27,33]. Regarding implementations of those, SPASS [33,43] is limited to a subset of the 15 logics, while LEO-III [13,36] supports all the logics in the modal cube, but can only solve very few of the available benchmark formulae.

KSP [23] is a modal logic theorem prover that implements both the modal-layered resolution (MLR) calculus [25] for the modal logic K and the global resolution (GMR) calculus [24] for all the 15 logics considered here. It also supports several refinements of resolution and a range of simplification rules. In this paper, we give reductions of all logics of the modal cube into a normal form for the basic modal logic K. We then compare the performance of the combination of these reductions with the modal-layered resolution calculus to that of the global resolution calculus on a new benchmark collection for the modal cube.

In [29] we have presented new reductions[1] of the propositional modal logics KB, KD, KT, K4, and K5 to Separated Normal Form with Sets of Modal Levels SNF_{sml}. SNF_{sml} is a generalisation of the Separated Normal Form with Modal Level, SNF_{ml}. In the latter, labelled modal clauses are used where a natural number label refers to a particular level within a tree Kripke structure at which a modal clause holds. In the former, a finite or infinite set of natural numbers labels each modal clause with the intended meaning that such a modal clause is true at every level of a tree Kripke structure contained in that set. As our prover KSP and the modal-layered resolution calculus it implements currently only support sets of modal clauses in SNF_{ml}, we then use a further reduction from SNF_{sml}

[1] A *reduction* here is a satisfiability preserving mapping between logics.

to SNF_{ml} to obtain an automatic theorem prover for these modal logics. Where all modal clauses are labelled with finite sets, this reduction is straightforward. This is the case for KB, KD and KT. For $\mathsf{K4}$ and $\mathsf{K5}$, characterised by the axioms $\Box\varphi \to \Box\Box\varphi$ and $\Diamond\varphi \to \Box\Diamond\varphi$, modal clauses are in general labelled with infinite sets. However, using a result by Massacci [21] for $\mathsf{K4}$ and an analogous result for $\mathsf{K5}$ by ourselves, we are able to bound the maximal level occurring in those labelling sets which in turn makes a reduction to SNF_{ml} possible.

Also in [29], we have shown experimentally that these reductions allow us to reason effectively in these logics, compared to the global modal resolution calculus [24] and to the relational and semi-functional translation built into the first-order theorem prover SPASS 3.9 [33,38,42]. The reason that the comparison only included a rather limited selection of provers is that these are the only ones with built-in support for all six logics our reductions covered.

Unfortunately, we cannot simply combine our reductions for single axioms to obtain satisfiability preserving reductions for their combinations. There are two main reasons for this. First, our calculus does not use an explicit representation of the accessibility relationship within a Kripke structure, which would make it possible to reflect modal axioms via corresponding properties of that accessibility relationship. Instead, we add labelled modal clauses based on instances of the modal axioms for $\Box$-formulae occurring in the modal formula we want to check for satisfiability. However, if we deal with multiple modal axioms, then these axioms might interact making it necessary to add instances that are not necessary for each individual axiom. For instance, consider, the converse of axiom B, $\Diamond\Box\varphi \to \varphi$, and axiom $\mathsf{4}$, $\Box\varphi \to \Box\Box\varphi$. Together they imply $\Diamond\Box\varphi \to \Box\varphi$. Instances of this derived axiom are necessary for completeness of a reduction from $\mathsf{KB4}$ to K, but are unsound for KB and $\mathsf{K4}$ separately.

Second, our reductions attempt to keep the labelling sets minimal in size in order to decrease the number of inferences that can be performed. Again, taking axioms B and $\mathsf{4}$ as examples, in KB, a $\Box$-formula $\Box\psi$ true at level ml in a tree-like Kripke structure M forces ψ to be true at level $ml - 1$, while in $\mathsf{K4}$, $\Box\psi$ true at level ml in M forces ψ to be true all levels ml' with $ml' > ml$. This is reflected in the labelling sets we use for these two logics. However, for $\mathsf{KB4}$, $\Box\psi$ true at level ml forces ψ to be true at every level in a tree-like Kripke structure M (unless M consists only of a single world).

Since we intend to maintain these two properties of our reductions, we have to consider each modal logic individually. As we will see, for some logics a reduction can be obtained as the union of the existing reductions while for others we need a logic-specific reduction to accommodate the interaction of axioms.

The structure of the paper is as follows. In Sect. 2 we recall common concepts of propositional modal logic and the definition of our normal form SNF_{ml}. Section 3 introduces our reduction for extensions of the basic modal logic K with combinations of the axioms B, D, T, $\mathsf{4}$, and $\mathsf{5}$. Section 4 presents a transformation from SNF_{sml} to SNF_{ml} which allows us to use the modal resolution prover KSP to reason in all the modal logics. In Sect. 5 we compare the performance of a combination of our reductions and the modal-layered resolution calculus implemented in the prover KSP with resolution calculi specifically designed for the logics under consideration as well as the prover LEO-III.

2 Preliminaries

The language of modal logic is an extension of the language of propositional logic with a unary modal operator $\Box$ and its dual $\Diamond$. More precisely, given a denumerable set of *propositional symbols*, $P = \{p, p_0, q, q_0, t, t_0, \ldots\}$ as well as propositional *constants* **true** and **false**, *modal formulae* are inductively defined as follows: constants and propositional symbols are modal formulae. If φ and ψ are modal formulae, then so are $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, $\Box\varphi$, and $\Diamond\varphi$. We also assume that $\wedge$ and $\vee$ are associative and commutative operators and consider, e.g., $(p \vee (q \vee r))$ and $(r \vee (q \vee p))$ to be identical formulae. We often omit parentheses if this does not cause confusion. By $\mathsf{var}(\varphi)$ we denote the set of all propositional symbols occurring in φ. This function straightforwardly extends to finite sets of modal formulae. A *modal axiom (schema)* is a modal formula ψ representing the set of all instances of ψ.

A *literal* is either a propositional symbol or its negation; the set of literals is denoted by L_P. By $\neg l$ we denote the *complement* of the literal $l \in L_P$, that is, if l is the propositional symbol p then $\neg l$ denotes $\neg p$, and if l is the literal $\neg p$ then $\neg l$ denotes p. By $|l|$ for $l \in L_P$ we denote p if $l = p$ or $l = \neg p$. A *modal literal* is either $\Box l$ or $\Diamond l$, where $l \in L_P$.

A *(normal) modal logic* is a set of modal formulae which includes all propositional tautologies, the axiom schema $\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$, called the *axiom* K, it is closed under modus ponens (if $\vdash \varphi$ and $\vdash \varphi \rightarrow \psi$ then $\vdash \psi$) and the rule of necessitation (if $\vdash \varphi$ then $\vdash \Box\varphi$).

K is the weakest modal logic, that is, the logic given by the smallest set of modal formulae constituting a normal modal logic. By $\mathsf{K}\Sigma$ we denote an *extension* of K by a set Σ of axioms.

The standard semantics of modal logics is the *Kripke semantics* or *possible world semantics*. A *Kripke frame* F is an ordered pair $\langle W, R\rangle$ where W is a non-empty set of *worlds* and R is a binary (accessibility) relation over W. A *Kripke structure* M over P is an ordered pair $\langle F, V\rangle$ where F is a Kripke frame and the *valuation* V is a function mapping each propositional symbol in P to a subset $V(p)$ of W. A *rooted Kripke structure* is an ordered pair $\langle M, w_0\rangle$ with $w_0 \in W$. To simplify notation, in the following we write $\langle W, R, V\rangle$ and $\langle W, R, V, w_0\rangle$ instead of $\langle\langle W, R\rangle, V\rangle$ and $\langle\langle\langle W, R\rangle, V\rangle, w_0\rangle$, respectively.

Satisfaction (or truth) of a formula at a world w of a Kripke structure $M = \langle W, R, V\rangle$ is inductively defined by:

$$\begin{array}{ll}
\langle M, w\rangle \models \textbf{true}; & \langle M, w\rangle \not\models \textbf{false}; \\
\langle M, w\rangle \models p & \text{iff } w \in V(p),\ \text{where } p \in P; \\
\langle M, w\rangle \models \neg\varphi & \text{iff } \langle M, w\rangle \not\models \varphi; \\
\langle M, w\rangle \models (\varphi \wedge \psi) & \text{iff } \langle M, w\rangle \models \varphi \text{ and } \langle M, w\rangle \models \psi; \\
\langle M, w\rangle \models (\varphi \vee \psi) & \text{iff } \langle M, w\rangle \models \varphi \text{ or } \langle M, w\rangle \models \psi; \\
\langle M, w\rangle \models (\varphi \rightarrow \psi) & \text{iff } \langle M, w\rangle \models \neg\varphi \text{ or } \langle M, w\rangle \models \psi; \\
\langle M, w\rangle \models \Box\varphi & \text{iff for every } v, w\, R\, v \text{ implies } \langle M, v\rangle \models \varphi; \\
\langle M, w\rangle \models \Diamond\varphi & \text{iff there is } v, w\, R\, v \text{ and } \langle M, v\rangle \models \varphi.
\end{array}$$

Table 1. Modal axioms and relational frame properties

Name	Axiom	Frame Property	
D	$\Box\varphi \rightarrow \Diamond\varphi$	Serial	$\forall v \exists w . v\, R\, w$
T	$\Box\varphi \rightarrow \varphi$	Reflexive	$\forall w . w\, R\, w$
B	$\varphi \rightarrow \Box\Diamond\varphi$	Symmetric	$\forall vw . v\, R\, w \rightarrow w\, R\, v$
4	$\Box\varphi \rightarrow \Box\Box\varphi$	Transitive	$\forall uvw . (u\, R\, v \wedge v\, R\, w) \rightarrow u\, R\, w$
5	$\Diamond\varphi \rightarrow \Box\Diamond\varphi$	Euclidean	$\forall uvw . (u\, R\, v \wedge u\, R\, w) \rightarrow v\, R\, w$

Table 2. Rewriting Rules for Simplification

$\varphi \wedge \varphi \Rightarrow \varphi$	$\varphi \wedge \neg\varphi \Rightarrow \mathbf{false}$	$\Box\mathbf{true} \Rightarrow \mathbf{true}$	$\neg\mathbf{true} \Rightarrow \mathbf{false}$	$\neg\neg\varphi \Rightarrow \varphi$
$\varphi \vee \varphi \Rightarrow \varphi$	$\varphi \vee \neg\varphi \Rightarrow \mathbf{true}$	$\Diamond\mathbf{false} \Rightarrow \mathbf{false}$	$\neg\mathbf{false} \Rightarrow \mathbf{true}$	
$\varphi \wedge \mathbf{true} \Rightarrow \varphi$	$\varphi \wedge \mathbf{false} \Rightarrow \mathbf{false}$	$\varphi \vee \mathbf{false} \Rightarrow \varphi$	$\varphi \vee \mathbf{true} \Rightarrow \mathbf{true}$	

If $\langle M, w\rangle \models \varphi$ holds then M is a *model* of φ, φ is *true at* w *in* M and M *satisfies* φ. A modal formula φ is *satisfiable* iff there exists a Kripke structure M and a world w in M such that $\langle M, w\rangle \models \varphi$.

We are interested in extensions of K with the modal axioms shown in Table 1 and their combinations. Each of these axioms defines a class of Kripke frames where the accessibility relation R satisfies the first-order property stated in the table. Combinations of axioms then define a class of Kripke frames where the accessibility relation satisfies the combination of their corresponding properties.

Given a normal modal logic L with corresponding class of frames $\mathfrak{F}$, we say a modal formula φ is *L-satisfiable* iff there exists a frame $F \in \mathfrak{F}$, a valuation V and a world $w \in F$ such that $\langle F, V, w\rangle \models \varphi$. It is *$L$-valid* or *valid in L* iff for every frame $F \in \mathfrak{F}$, every valuation V and every world $w \in F$, $\langle F, V, w\rangle \models \varphi$. A normal modal logic L_2 is *an extension* of a normal modal logic L_1 iff all L_1-valid formulae are also L_2-valid.

A rooted Kripke structure $M = \langle W, R, V, w_0\rangle$ is a *rooted tree Kripke structure* iff R is a tree, that is, a directed acyclic connected graph where each node has at most one predecessor, with *root* w_0. It is a *rooted tree Kripke model* of a modal formula φ iff $\langle W, R, V, w_0\rangle \models \varphi$. In a rooted tree Kripke structure with root w_0 for every world $w_k \in W$ there is exactly one path connecting w_0 and w_k, the length of that path is the *modal level of* w_k *(in M), denoted by* $\mathsf{ml}_M(w_k)$.

It is well-known [17] that a modal formula φ is K-satisfiable iff there is a finite rooted tree Kripke structure $M = \langle F, V, w_0\rangle$ such that $\langle M, w_0\rangle \models \varphi$.

For the reductions presented in the next section we assume that any modal formula φ has been simplified by exhaustively applying the rewrite rules in Table 2, and it is in Negation Normal Form (NNF). That is, a formula where only propositional symbols are allowed in the scope of negations. We say that such a formula is in *simplified NNF*.

The reductions produce formulae in a clausal normal form, called *Separated Normal Form with Sets of Modal Levels* SNF_{sml}, introduced in [29]. The language of SNF_{sml} extends that of the basic modal logic K with sets of modal levels as labels. Clauses in SNF_{sml} have one of the following forms:

$S : \bigvee_{i=1}^{n} l_i$	$S : l' \rightarrow \Box l$	$S : l' \rightarrow \Diamond l$
(literal clause)	(positive modal clause)	(negative modal clause)

where $S \subseteq \mathbb{N}$ and l, l', l_i are propositional literals with $1 \leq i \leq n$, $n \in \mathbb{N}$. We write $\star : \varphi$ instead of $\mathbb{N} : \varphi$ and such clauses are called *global clauses*. Positive and negative modal clauses are together known as *modal clauses*.

Given a rooted tree Kripke structure M and a set S of natural numbers, by $M[S]$ we denote the set of worlds that are at a modal level in S, that is, $M[S] = \{w \in W \mid \mathsf{ml}_M(w) \in S\}$. Then

$$M \models S : \varphi \text{ iff } \langle M, w \rangle \models \varphi \text{ for every world } w \in M[S].$$

The motivation for using a set S to label clauses is that in our reductions the formula φ may hold at several levels, possibly an infinite number of levels. It therefore makes sense to label such formulae not with just a single level, but a set of levels. The Separated Normal Form with Modal Level, SNF_{ml}, can be seen as the special case of SNF_{sml} where all labelling sets are singletons.

Note that if $S = \emptyset$, then $M \models S : \varphi$ trivially holds. Also, a Kripke structure M can satisfy S : **false** if there is no world w with $\mathsf{ml}_M(w) \in S$. On the other hand, S : **false** with $0 \in S$ is unsatisfiable as a rooted tree Kripke structure always has a world with modal level 0.

If $M \models S : \varphi$, then we say that $S : \varphi$ *holds in* M or *is true in* M. For a set Φ of labelled formulae, $M \models \Phi$ iff $M \models S : \varphi$ for every $S : \varphi$ in Φ, and we say Φ is K-*satisfiable*.

We introduce some notation that will be used in the following. Let $S^{+} = \{l+1 \in \mathbb{N} \mid l \in S\}$, $S^{-} = \{l-1 \in \mathbb{N} \mid l \in S\}$, and $S^{\geq} = \{n \mid n \geq \min(S)\}$, where $\min(S)$ is the least element in S. Note that the restriction of the elements being in $\mathbb{N}$ implies that S^{-} cannot contain negative numbers.

3 Extensions of K

In this section we define reductions from all the logics in the modal cube to SNF_{sml}. We assume that the set P of propositional symbols is partitioned into two infinite sets Q and T such that Q contains the propositional symbols of the modal formula φ under consideration, and T surrogate symbols t_ψ for every subformula ψ of φ and supplementary propositional symbols. In particular, for every modal formula ψ we have $\mathsf{var}(\psi) \subset Q$ and there exists a propositional symbol $t_\psi \in T$ uniquely associated with ψ. These surrogate symbols serve the same purpose as Tseitin variables [40] and Skolem predicates [30,39] in the transformation of propositional and first-order formulae, respectively, to clausal form via structural transformation.

It turns out that given a reduction $\rho_{\mathsf{K}\Sigma}$ for $\mathsf{K}\Sigma$ with $\{\mathsf{D}, \mathsf{T}\} \cap \Sigma = \emptyset$, there is a uniform and straightforward way we can obtain a reduction for $\mathsf{KD}\Sigma$ and $\mathsf{KT}\Sigma$ from $\rho_{\mathsf{K}\Sigma}$. Also, the valid formulae of $\mathsf{KDT}\Sigma$ are the same as those of

Table 3. Categorisation of modal logics in the modal cube

‘Base logics’	K	KB	K4	K5	KB4	K45
Extensions with D	KD	KDB	KD4	KD5		KD45
Extensions with T	KT	KTB	KT4	KT5		

$\mathsf{KT}\Sigma$, so we do not need to consider the case of adding both axioms to $\mathsf{K}\Sigma$. Similarly, the logics KT45, KDB4, KTB4 and KT5 all have the same set of valid formulae. Therefore, as shown in Table 3, we can divide the 15 modal logics into three categories: Six ‘base logics’, five modal logics obtained by extending a ‘base logic’ with D, and a further four modal logics obtained by extending a ‘base logic’ with T. For four of the six ‘base logics’ (namely, K, KB, K4, and K5) we have already devised reductions in [29], so only two (i.e., KB4 and K45) remain.

Given a modal formula φ in simplified NNF and $L = \mathsf{K}\Sigma$ with $\Sigma \subseteq \{\mathsf{B}, \mathsf{D}, \mathsf{T}, \mathsf{4}, \mathsf{5}\}$, we can obtain a set Φ_L of clauses in SNF_{sml} such that φ is L-satisfiable iff Φ_L is K-satisfiable with $\Phi_L = \rho_L^{sml}(\varphi) = \{\{0\} : t_\varphi\} \cup \rho_L(\{0\} : t_\varphi \to \varphi)$, where ρ_L is defined as follows:

$$
\begin{aligned}
\rho_L(S : t \to \mathbf{true}) &= \emptyset\\
\rho_L(S : t \to \mathbf{false}) &= \{S : \neg t\}\\
\rho_L(S : t \to (\psi_1 \wedge \psi_2)) &= \{S : \neg t \vee \eta(\psi_1), S : \neg t \vee \eta(\psi_2)\} \cup \delta_L(S, \psi_1) \cup \delta_L(S, \psi_2)\\
\rho_L(S : t \to \psi) &= \{S : \neg t \vee \psi\}\\
&\quad\text{if } \psi \text{ is a disjunction of literals}\\
\rho_L(S : t \to (\psi_1 \vee \psi_2)) &= \{S : \neg t \vee \eta(\psi_1) \vee \eta(\psi_2)\} \cup \delta_L(S, \psi_1) \cup \delta_L(S, \psi_2)\\
&\quad\text{if } \psi_1 \vee \psi_2 \text{ is not a disjunction of literals}\\
\rho_L(S : t \to \Diamond\psi) &= \{S : t \to \Diamond\eta(\psi)\} \cup \delta_L(S^+, \psi)\\
\rho_L(S : t \to \Box\psi) &= P_L(S : t \to \Box\psi) \cup \Delta_L(S : t \to \Box\psi)
\end{aligned}
$$

η and δ_L are defined as follows:

$$
\eta(\psi) = \begin{cases} \psi, & \text{if } \psi \text{ is a literal}\\ t_\psi, & \text{otherwise} \end{cases}
\qquad
\delta_L(S, \psi) = \begin{cases} \emptyset, & \text{if } \psi \text{ is a literal}\\ \rho_L(S : t_\psi \to \psi), & \text{otherwise} \end{cases}
$$

and functions P_L and Δ_L, are defined as shown in Table 4.

We can see in Table 4 that the reduction for KB4 has an additional SNF_{sml} clause $\star : t_{\Box\psi} \vee t_{\Box\neg t_{\Box\psi}}$ that occurs neither in the reduction for KB nor in that for K4. It can be seen as an encoding of the derived axiom $\Diamond\Box\psi \to \Box\psi$ that follows from the contrapositive $\Diamond\Box\psi \to \psi$ of B and 4 $\Box\psi' \to \Box\Box\psi'$.

For K45 we see that all the SNF_{sml} clauses in the reduction for K5 carry over. These clauses are already sufficient to ensure that, semantically, if $t_{\Box\psi}$ is true at any world at a level other than 0, then $t_{\Box\psi}$ is true at every world. Consequently, to accommodate axiom 4, it suffices to add the SNF_{sml} clause $\{0\} : t_{\Box\psi} \to \Box t_{\Box\psi}$ to ensure that this also holds for the root world at level 0.

L	$P_L(S : t_{\Box\psi} \rightarrow \Box\psi)$	$\Delta_L(S : t_{\Box\psi} \rightarrow \Box\psi)$
K	$S : t_{\Box\psi} \rightarrow \Box\eta(\psi)$	$\delta_L(S^+, \psi)$
KB	$S : t_{\Box\psi} \rightarrow \Box\eta(\psi)$, $S^- : \eta(\psi) \vee t_{\Box\neg t_{\Box\psi}}$, $S^- : t_{\Box\neg t_{\Box\psi}} \rightarrow \Box\neg t_{\Box\psi}$	$\delta_L(S^- \cup S^+, \psi)$
K4	$S^{\geq} : t_{\Box\psi} \rightarrow \Box\eta(\psi)$, $S^{\geq} : t_{\Box\psi} \rightarrow \Box t_{\Box\psi}$	$\delta_L((S^+)^{\geq}, \psi)$
K5	$\star : t_{\Box\psi} \rightarrow \Box\eta(\psi)$, $\star : \neg t_{\Diamond t_{\Box\psi}} \vee t_{\Box\psi}$, $\star : t_{\Diamond t_{\Box\psi}} \rightarrow \Diamond t_{\Box\psi}$, $\star : \neg t_{\Diamond t_{\Box\psi}} \rightarrow \Box\neg t_{\Box\psi}$, $\star : t_{\Diamond t_{\Box\psi}} \rightarrow \Box t_{\Diamond t_{\Box\psi}}$	$\delta_L(\star, \psi)$
KB4	$\star : t_{\Box\psi} \rightarrow \Box\eta(\psi)$, $\star : \eta(\psi) \vee t_{\Box\neg t_{\Box\psi}}$, $\star : t_{\Box\psi} \vee t_{\Box\neg t_{\Box\psi}}$, $\star : t_{\Box\neg t_{\Box\psi}} \rightarrow \Box\neg t_{\Box\psi}$, $\star : t_{\Box\psi} \rightarrow \Box t_{\Box\psi}$	$\delta_L(\star, \psi)$
K45	$\star : t_{\Box\psi} \rightarrow \Box\eta(\psi)$, $\{0\} : t_{\Box\psi} \rightarrow \Box t_{\Box\psi}$ iff $0 \in S$, $\star : \neg t_{\Diamond t_{\Box\psi}} \vee t_{\Box\psi}$, $\star : t_{\Diamond t_{\Box\psi}} \rightarrow \Diamond t_{\Box\psi}$, $\star : \neg t_{\Diamond t_{\Box\psi}} \rightarrow \Box\neg t_{\Box\psi}$, $\star : t_{\Diamond t_{\Box\psi}} \rightarrow \Box t_{\Diamond t_{\Box\psi}}$	$\delta_L(\star, \psi)$
KDΣ	$\{lb^P_{\mathsf{K}\Sigma}(S) : t_{\Box\psi} \rightarrow \Diamond\eta(\psi)\} \cup P_{\mathsf{K}\Sigma}(S : t_{\Box\psi} \rightarrow \Box\psi)$	$\delta_L(lb^\delta_{\mathsf{K}\Sigma}(S), \psi)$
KTΣ	$\{lb^P_{\mathsf{K}\Sigma}(S) : \neg t_{\Box\psi} \vee \eta(\psi)\} \cup P_{\mathsf{K}\Sigma}(S : t_{\Box\psi} \rightarrow \Box\psi)$	$\delta_L(lb^\delta_{\mathsf{K}\Sigma}(S) \cup S, \psi)$

where $lb^P_{\mathsf{K}\Sigma}$ and $lb^\delta_{\mathsf{K}\Sigma}$ are defined as follows

Table 4. Reduction of $\Box$-formulae, $\Sigma \subseteq \{\mathsf{B}, \mathsf{4}, \mathsf{5}\}$.

L	K	KB	K4	K5	KB4	K45
$lb^P_L(S)$	S	S	$S^{\geq}$	$\star$	$\star$	$\star$
$lb^\delta_L(S)$	S^+	$S^- \cup S^+$	$(S^+)^{\geq}$	$\star$	$\star$	$\star$

For reductions of KDΣ and KTΣ we have favoured the reuse of reductions for KΣ, KD and KT over optimisation for specific logics. For example, take KBD. Given that in a symmetric model, every world w except the root world w_0 has an R-successor, the axiom D only 'enforces' that w_0 also has an R-successor. So, instead of adding a clause $S : t_{\Box\psi} \rightarrow \Diamond\psi$ for every clause $S : t_{\Box\psi} \rightarrow \Box\eta(\psi)$ we could just add $\{0\} : t_{\Box\psi} \rightarrow \Diamond\psi$ iff $0 \in S$. Similarly, in KT5, because of 5, for all worlds w except w_0 we already have $w \, R \, w$. So, we could again $\{0\} : \neg t_{\Box\psi} \vee \eta(\psi)$ for every clause $S : t_{\Box\psi} \rightarrow \Box\eta(\psi)$ iff $0 \in S$.

For the KB4-unsatisfiable formula $\psi_1 = (\neg p \wedge \Diamond\Diamond\Box p)$, if we were to independently apply the reductions for KB and K4, that is, we compute $\{\{0\} : t_{\psi_1}\} \cup \rho_{\mathsf{KB}}(\{0\} : t_{\psi_1} \rightarrow \psi_1) \cup \rho_{\mathsf{K4}}(\{0\} : t_{\psi_1} \rightarrow \psi_1)$, then the result is the following set of clauses Φ_1:

(1) $\{0\} : t_{\psi_1}$
(2) $\{0\} : \neg t_{\psi_1} \vee \neg p$
(3) $\{0\} : \neg t_{\psi_1} \vee t_{\Diamond\Diamond\Box p}$
(4) $\{0\} : t_{\Diamond\Diamond\Box p} \rightarrow \Diamond t_{\Diamond\Box p}$
(5) $\{1\} : t_{\Diamond\Box p} \rightarrow \Diamond t_{\Box p}$
(6) $\{2\}^{\geq} : t_{\Box p} \rightarrow \Box p$
(7) $\{2\}^{\geq} : t_{\Box p} \rightarrow \Box t_{\Box p}$
(8) $\{1\} : p \vee t_{\Box\neg t_{\Box p}}$
(9) $\{1\} : t_{\Box\neg t_{\Box p}} \rightarrow \Box\neg t_{\Box p}$

Clauses (1) to (5) stem from the transformation of ψ_1 to SNF_{sml} for K, Clauses (6) and (7) stem from the reduction for 4 and Clauses (8) and (9) stem from the reduction for B. This set of SNF_{sml} clauses is K-satisfiable. The clauses imply $\{1\} : p$, but neither $\{1\} : \Box p$ nor $\{0\} : p$ which we need to obtain a contradiction. Part of the reason is that we would need to apply the reduction for $\mathsf{4}$ and B recursively to newly introduced surrogates for $\Box$-formulae which in turn leads to the introduction of further surrogates and problems with the termination of the reduction.

In contrast, the clause set Φ_2 obtained by our reduction for $\mathsf{KB4}$ is:

(10) $\{0\} : t_{\psi_1}$
(11) $\{0\} : \neg t_{\psi_1} \vee \neg p$
(12) $\{0\} : \neg t_{\psi_1} \vee t_{\Diamond\Diamond\Box p}$
(13) $\{0\} : t_{\Diamond\Diamond\Box p} \rightarrow \Diamond t_{\Diamond\Box p}$
(14) $\{1\} : t_{\Diamond\Box p} \rightarrow \Diamond t_{\Box p}$
(15) $\star : t_{\Box p} \rightarrow \Box p$
(16) $\star : t_{\Box p} \rightarrow \Box t_{\Box p}$
(17) $\star : p \vee t_{\Box\neg t_{\Box p}}$
(18) $\star : t_{\Box\neg t_{\Box p}} \rightarrow \Box\neg t_{\Box p}$
(19) $\star : t_{\Box p} \vee t_{\Box\neg t_{\Box p}}$
(20) $\star : t_{\Box\neg t_{\Box p}} \rightarrow \Box t_{\Box\neg t_{\Box p}}$

Note Clauses (19) and (20) in Φ_2 for which there are no corresponding clauses in Φ_1. Also, the set of labels of Clauses (15) to (18) are strict supersets of those of the corresponding Clauses (6) to (9). Φ_2 implies both $\{1\} : \Box p$ and $\{0\} : p$. The latter, together with Clauses (10) and (11), means Φ_2 is K-unsatisfiable.

Theorem 1. *Let φ be a modal formula in simplified NNF, $\Sigma \subseteq \{\mathsf{B}, \mathsf{D}, \mathsf{T}, \mathsf{4}, \mathsf{5}\}$, and $\Phi_{\mathsf{K}\Sigma} = \rho^{sml}_{\mathsf{K}\Sigma}(\varphi)$. Then φ is $\mathsf{K}\Sigma$-satisfiable iff $\Phi_{\mathsf{K}\Sigma}$ is K-satisfiable.*

Proof (Sketch). For $|\Sigma| \leq 1$ this follows from Theorem 5 in [29].

For $\mathsf{K45}$, $\mathsf{KB4}$, $\mathsf{KD}\Sigma'$, and $\mathsf{KT}\Sigma'$ with $\Sigma' \subseteq \{\mathsf{B}, \mathsf{4}, \mathsf{5}\}$ we proceed in analogy to the proofs of Theorems 3 and 4 in [29]. Let L be one of these logics.

To show that if φ is L-satisfiable then Φ_L is K-satisfiable, we show that given a rooted L-model M of φ a small variation of the unravelling of M is a rooted tree K-model $\vec{M}_L$ of Φ_L. The main step is to define the valuation of the additional propositional symbols t_ψ so that we can prove that all clauses in Φ_L hold in $\vec{M}_L$. To show that if Φ_L is K-satisfiable then φ is L-satisfiable, we take a rooted tree K-model $M = \langle W, R, V, w_0 \rangle$ of Φ_L and construct a Kripke structure $M_L = \langle W, R^L, V, w_0 \rangle$. The relation R^L is the closure of R under the relational properties associated with the axioms of L. The proof that M_L is a model of φ relies on the fact that the clauses in Φ_L ensure that for subformulae $\Box\psi$ of φ, ψ will be true at all worlds reachable via R^L from a world where $\Box\psi$ is true. □

4 From SNF_{sml} to SNF_{ml}

As K$_\mathsf{S}$P does not support SNF_{sml}, in our evaluation of the effectiveness of the reductions defined in Sect. 3, we have used a transformation from SNF_{sml} to SNF_{ml}. An alternative approach would be to reflect the use of SNF_{sml} in the calculus and re-implement the prover. Whilst we believe that redesigning the calculus presents few problems, re-implementing K$_\mathsf{S}$P needs more thought in particular how to represent infinite sets. The route we adopt here allows us to experiment with the approach in general without having to change the prover. For extensions of K with one or more of the axioms B, D, T such a transformation

Table 5. Bounds on the length of prefixes in SST tableaux

Logic L	Bound db^{φ}_{L}
K,KD,KT, KB,KDB,KTB	$1 + d^{\varphi}_{m}$
K4,S4	$2 + d^{\varphi}_{\Diamond} + n^{\varphi}_{\Diamond} \times n^{\varphi}_{\Box}$
KD4	$2 + d^{\varphi}_{\Diamond} + (\max(1, n^{\varphi}_{\Diamond}) \times n^{\varphi}_{\Box})$
KB4,KTB4, K5,S5,K45	$2 + d^{\varphi}_{\Diamond} + n^{\varphi}_{\Diamond}$
KD5	$2 + d^{\varphi}_{\Diamond} + \max(1, n^{\varphi}_{\Diamond})$

is straightforward as the sets of modal levels occurring in the normal form of modal formulae are all finite. Thus, instead of a single SNF_{sml} clause $S : \neg t_{\psi} \vee \eta_{f}(\psi)$ we can use the finite set of SNF_{ml} clauses $\{ml : \neg t_{\psi} \vee \eta_{f}(\psi) \mid ml \in S\}$.

For extensions of K with at least one of the axioms $\mathsf{4}$ and $\mathsf{5}$, potentially together with other axioms, the sets of modal levels labelling clauses are in general infinite. For each logic L it is, however, possible to define a computable function that maps the modal formula φ under consideration onto a bound db^{φ}_{L} such that, restricting the modal levels in the normal form of φ by db^{φ}_{L}, preserves satisfiability equivalence.

To establish the bound and prove satisfiability equivalence, we need to introduce the basic notions of Single Step Tableaux (SST) calculi for a modal logic L [14,21], which uses sequences of natural numbers to prefix modal formulae in a tableau. The SST calculus consists of a set of rules, with the (π) rule being the only rule increasing prefixes' lengths (i.e., $\sigma : \Diamond\varphi/\sigma.n : \varphi$ with $\sigma.n$ new on the branch). For a logic L, an *L-tableau* $\mathcal{T}$ in the SST calculus for a modal formula φ is a (binary) tree where the root of $\mathcal{T}$. is labelled with $1 : \varphi$, and every other node is labelled with a prefixed formula $\sigma : \psi$ obtained by application of a rule of the calculus. A *branch* $\mathcal{B}$ is a path from the root to a leaf. A *branch* $\mathcal{B}$ is *closed* if it contains either **false** or a propositional contradiction at the same prefix. A *tableau* "$\mathcal{T}$ is *closed* if all its branches are closed. A prefixed formula $\sigma : \psi$ is *reduced for rule* (r) *in* $\mathcal{B}$ if the branch $\mathcal{B}$ already contains the conclusion of such rule application. By a *systematic tableau construction* we mean an application of the procedure in [14, p. 374] adapted to SST rules.

For each logic L, we establish its bound by considering an L-SST calculus, where a modal level in an SNF_{sml} clause corresponds to the length of a prefix in an SST tableau. The bound then either follows from an already known bound on the length of prefixes in an SST tableau preserving correctness of the SST calculus, or we establish such a bound ourselves. To prove satisfiability equivalence, we show that, for a closed SST tableau with such a bound on the length of prefixes in place, we can construct a resolution refutation of a set of SNF_{sml} or SNF_{ml} clauses with a corresponding bound on modal levels in those clauses.

For a modal formula φ in simplified NNF let d^{φ}_{m} be the modal depth of φ, $d^{\varphi}_{\Diamond}$ be the maximal nesting of $\Diamond$-operators not under the scope of any $\Box$ operators in φ, $n^{\varphi}_{\Box}$ be the number of $\Box$-subformulae in φ, and $n^{\varphi}_{\Diamond}$ be the number of

$\Diamond$-subformulae below $\Box$-operators in φ. Our results for the bounds on the length of prefixes in SST tableaux can then be summarised by the following theorem.

Theorem 2. *Let $L = \mathsf{K}\Sigma$ with $\Sigma \subseteq \{\mathsf{B}, \mathsf{D}, \mathsf{T}, \mathsf{4}, \mathsf{5}\}$. A systematic tableau construction of an L-tableau for a modal formula φ in simplified NNF under the following Constraints (TC1) and (TC2)*

(TC1) a rule (r) of the SST calculus is only applicable to a prefixed formula $\sigma : \psi$ in a branch $\mathcal{B}$ if the formula is not already reduced for (r) in $\mathcal{B}$;
(TC2) rule (π) of the SST calculus is only applicable to prefixed formulae $\sigma : \Diamond\psi$ with $|\sigma| < db^{\varphi}_{L}$ for db^{φ}_{L} as defined in Table 5

terminates in one of following states:

(1) all branches of the constructed tableau are closed and φ is L-unsatisfiable or
(2) at least one branch $\mathcal{B}$ is not closed, no rule is still applicable to a labelled formula in $\mathcal{B}$, and φ is L-satisfiable.

The proof is analogous to Massacci's [21, Section B.2]. Note that for logics $\mathsf{KD4}$ and $\mathsf{KD5}$, we use $\max(1, n^{\varphi}_{\Diamond})$ in the calculation of the bound. That is, if $n^{\varphi}_{\Diamond} \geq 1$ then $\max(1, n^{\varphi}_{\Diamond}) = n^{\varphi}_{\Diamond}$ and the bound is the same as for $\mathsf{K4}$ and $\mathsf{K5}$. Otherwise $\max(1, n^{\varphi}_{\Diamond}) = 1$, that is, the bound is the same as for a formula with a single $\Diamond$-subformula below $\Box$-operators in φ.

For K, KD, KT, KB and KDB these bounds were already stated in [21, Tables III and IV]. The bound for KTB follows straightforwardly from that for KB and KDB. For $\mathsf{KD4}$, Massacci [21, Tables III and IV] states the bound to be the same as for $\mathsf{K4}$. However, this is not correct for the case that the formula φ contains no $\Diamond$-formulae, where its bound would simply be 2, independent of φ. For example, the formula $\Box\Box\Box\mathbf{false}$ which is $\mathsf{KD4}$-unsatisfiable, does not have a closed $\mathsf{KD4}$-tableau with this bound. For the other logics the bounds are new. As argued in [21], the bounds allow tableau decision procedures for extensions of K with axioms $\mathsf{4}$ and $\mathsf{5}$ that do not require a loop check and are therefore of wider interest.

Note that in $\mathsf{KT4}$, $\Box\Box\psi$ and $\Box\psi$ are equivalent and so are $\Box(\psi \wedge \Box\vartheta)$ and $\Box(\psi \wedge \vartheta)$. So, it makes sense to further simplify $\mathsf{KT4}$ formulae using such equivalences before computing the normal form and the bound with the benefit that it may not only reduce the bound but also the size of the normal form. Similar equivalences that can be used to reduce the number of modal operators in a formula also exist for other logics, see, e.g., [8, Chapter 4].

To establish a relationship between closed tableaux and resolution refutations of a set of SNF_{ml} clauses, we formally define the modal layered resolution calculus. Table 6 shows the inference rules of the calculus restricted to labels occurring in our normal form. For GEN1 and GEN3, if the modal clauses in the premises occur at the modal level ml, then the literal clause in the premises occurs at modal level $ml + 1$.

Let Φ be a set of SNF_{ml} clauses. A *(resolution) derivation from* Φ is a sequence of sets $\Phi_0, \Phi_1, \ldots$ where $\Phi_0 = \Phi$ and, for each $i > 0$, $\Phi_{i+1} = \Phi_i \cup \{D\}$, where $D \notin \Phi_i$ is the resolvent obtained from Φ_i by an application of one of the inference rules to premises in Φ_i. A *(resolution) refutation of* Φ is a derivation $\Phi_0, \ldots, \Phi_k$, $k \in \mathbb{N}$, where $0 : \mathbf{false} \in \Phi_k$.

To map a set of SNF_{sml} clauses to a set of SNF_{ml} clauses, using a bound $n \in \mathbb{N}$ on the modal levels, we define a function db_n on clauses and sets of clauses in SNF_{sml} as follows:

$$\begin{aligned} \mathrm{db}_n(S : \varphi) &= \{ml : \varphi \mid ml \in S \text{ and } ml \leq n\} \\ \mathrm{db}_n(\Phi) &= \textstyle\bigcup_{S:\varphi \in \Phi} \mathrm{db}_n(S : \varphi) \end{aligned}$$

Note that prefixes in SST-tableaux have a minimal length of 1 while the minimal modal level in SNF_{ml} clauses is 0. So, a prefix of length n in a prefixed formula corresponds to a modal level $n - 1$ in an SNF_{ml} clause.

The proof of the following theorem then takes advantage of the fact that we have surrogates and associated clauses for each subformula of φ and proceeds by induction over applications of rule (π).

Theorem 3. *Let $L = \mathsf{K}\Sigma$ with $\Sigma \subseteq \{\mathsf{B}, \mathsf{D}, \mathsf{T}, \mathsf{4}, \mathsf{5}\}$, φ be a $\mathsf{K}\Sigma$-unsatisfiable formula in simplified NNF, db^{φ}_{L} be as defined in Table 5, and $\Phi_L = \rho^{ml}_{L}(\varphi) = \mathrm{db}_{db^{\varphi}_{L}-1}(\rho^{sml}_{L}(\varphi))$. Then there is a resolution refutation of Φ_L.*

Regarding the size of the encoding, we note that, ignoring the labelling sets, the reduction ρ^{sml}_{L} into SNF_{sml} is linear with respect to the size of the original formula. The size including the labelling sets would depend on the exact representation of those sets, in particular, of infinite sets. As those are not arbitrary, there is still an overall polynomial bound on the size of the sets of SNF_{sml} clauses produced by ρ^{sml}_{L}. When transforming clauses from SNF_{sml} into SNF_{ml}, we may need to add every clause to all levels within the bounds provided by Theorem 3. The parameters for calculating those bounds, d^{φ}_{m}, $d^{\varphi}_{\Diamond}$, $n^{\varphi}_{\Diamond}$, and $n^{\varphi}_{\Box}$, are all themselves linearly bound by the size of the formula. Thus, in the worst case, which is $\mathsf{S4}$, the size of the clause set produced by ρ^{ml}_{L} is bounded by a polynomial of degree 3 with respect to the size of the original formula.

It is worth pointing out that both the reduction ρ^{sml}_{L} of a modal formula to SNF_{sml} and the reduction ρ^{ml}_{L} to SNF_{ml} are also reversible, that is, we can reconstruct the original formula from the SNF_{sml} and from the SNF_{ml} clause set obtained by ρ^{sml}_{L} or ρ^{ml}_{L}, respectively. This reconstruction can also be performed in polynomial time. Thus the reduction itself does not affect the complexity of the satisfiability problem. For instance, the satisfiability problem for $\mathsf{S5}$ is NP-complete and so is the satisfiability problem of the subclass $\mathbb{C}_{\mathsf{S5}}$ of SNF_{ml} clause sets that can be obtained as the result of an application of ρ^{ml}_{S5} to a modal formula. However, a generic decision procedure for K will not be a complexity-optimal decision procedure for $\mathbb{C}_{\mathsf{S5}}$.

Table 6. Inference rules of the MLR calculus

$$\mathsf{LRES}: \frac{\begin{array}{c} ml: D \vee l \\ ml: D' \vee \neg l \end{array}}{ml: D \vee D'} \qquad \mathsf{MRES}: \frac{\begin{array}{c} ml: l_1 \rightarrow \Box l \\ ml: l_2 \rightarrow \Diamond \neg l \end{array}}{ml: \neg l_1 \vee \neg l_2} \qquad \mathsf{GEN2}: \frac{\begin{array}{c} ml: l'_1 \rightarrow \Box l_1 \\ ml: l'_2 \rightarrow \Box \neg l_1 \\ ml: l'_3 \rightarrow \Diamond l_2 \end{array}}{ml: \neg l'_1 \vee \neg l'_2 \vee \neg l'_3}$$

$$\mathsf{GEN1}: \frac{\begin{array}{c} ml: l'_1 \rightarrow \Box \neg l_1 \\ \vdots \\ ml: l'_m \rightarrow \Box \neg l_m \\ ml: l' \rightarrow \Diamond \neg l \\ ml+1: l_1 \vee \ldots \vee l_m \vee l \end{array}}{ml: \neg l'_1 \vee \ldots \vee \neg l'_m \vee \neg l'} \qquad \mathsf{GEN3}: \frac{\begin{array}{c} ml: l'_1 \rightarrow \Box \neg l_1 \\ \vdots \\ ml: l'_m \rightarrow \Box \neg l_m \\ ml: l' \rightarrow \Diamond l \\ ml+1: l_1 \vee \ldots \vee l_m \end{array}}{ml: \neg l'_1 \vee \ldots \vee \neg l'_m \vee \neg l'}$$

5 Evaluation

An empirical evaluation of the practical usefulness of the reductions we presented in Sects. 3 and 4 faces the challenge that there is no substantive collection of benchmark formulae for the 15 logics of the modal cube except for basic modal logic. Catach [7] evaluates his prover on 31 modal formulae with a maximal length of 22 and maximal modal depth of 4. They are not sufficiently challenging. The QMLTP Problem Library for First-Order Modal Logics [32] focuses on quantified formulae and contains only a few formulae taken from the research literature that are purely propositional and were not written for the basic modal logic K. The Logics Workbench (LWB) benchmark collection [2] contains formulae for K, KT and S4 but not for any of the other logics we consider. For each of these three logics, the collection consists of 18 parameterised classes with 21 formulae each, plus scripts with which further formulae could be generated if needed. All formulae in 9 classes are satisfiable and all formulae in the other 9 classes are unsatisfiable in the respective logic.

In [29] we have used the 18 classes of the LWB benchmark collection for K to evaluate our approach for the six logics consisting of K and its extensions with a single axiom. One drawback of using these 18 classes for other modal logics is that formulae that are K-satisfiable are not necessarily KΣ-satisfiable for non-empty sets Σ of additional axioms. For example, for K5, only 60 out of 180 K-satisfiable formulae were K5-satisfiable. Another drawback is that while K-unsatisfiable formulae are also KΣ-unsatisfiable, a resolution refutation would not necessarily involve any of the additional clauses introduced by our reduction for KΣ. It may be that the additional clauses allow us to find a shorter refutation, but it may just be a case of finding the same refutation in a larger search space. It is also worth recalling that simplification alone is sufficient to determine that all formulae in the class `k_lin_p` are K-unsatisfiable while pure literal elimination can be used to reduce all formulae in `k_grz_p` to the same simple formula [26].

Table 7. Logic-specific modification of unsatisfiable benchmark formulae

Logic L	ψ_l^p	Logic L	ψ_l^p
K	**false**	KD4	$(\Box q_p \wedge \Diamond\Diamond\Box\Diamond\neg q_p)$
KB	$(\neg q_p \wedge \Diamond\Box q_p)$	K5	$(\Diamond\neg q_p \wedge \Diamond\Box q_p)$
KDB	$(\neg q_p \wedge \Diamond\Box((\Box\neg q'_p \wedge \Box q'_p) \vee q_p))$	KD5	$((\Box\neg q_p \wedge \Box q_p) \vee (\Diamond\Box q'_p \wedge \Diamond\neg q'_p)$
KTB	$(\neg q_p \wedge \Diamond\Box((\neg q'_p \wedge \Box q'_p) \vee q_p))$	K45	$(\Box q_p \wedge \Diamond\Box q'_p \wedge \Diamond\Diamond(\neg q_p \vee \neg q'_p))$
KD	$(\Box\neg q_p \wedge \Box q_p)$	KD45	$((\Box\neg q'_p \wedge \Box q'_p) \wedge$
KT	$(\neg q_p \wedge \Box q_p)$		$(\Box q_p \wedge \Diamond\Box q'_p \wedge \Diamond\Diamond(\neg q_p \vee \neg q'_p))$
K4	$(\Box q_p \wedge \Diamond\Diamond\neg q_p)$	S4	$(\neg q'_p \wedge \Box(\neg q'_p \vee \Box q_p) \wedge \Diamond\Diamond\neg q_p)$
K4B	$(\neg q_p \wedge \Diamond\Diamond\Box q_p)$	S5	$((\neg q_p \wedge \Box q_p) \vee (\neg q'_p \wedge \Diamond\Diamond\Diamond\Box q'_p)$

Thus, some of the classes evaluate the preprocessing capabilities of a prover but not the actual calculus and its implementation.

We therefore propose a different approach here. The principles underlying our approach are that (i) there should be the same number of formulae for each logic though not necessarily the same formulae across all logics; (ii) there should be an equal number of satisfiable and unsatisfiable formulae for each logic; (iii) a formula that is L-unsatisfiable should only be L'-unsatisfiable for every extension L' of L; (iv) a formula that is L'-satisfiable should be L-satisfiable for every extension L' of L; (v) the formulae should belong to parameterised classes of formulae of increasing difficulty. Note that Principles (iii) and (iv) are intentionally not symmetric. For L-unsatisfiable formulae it should be necessary for a prover to use the rules or clauses specific to L instead of being able to find a refutation without those. For L-satisfiable formulae we want to maximise the search space for a model.

For unsatisfiable formulae, we take the five LWB classes `k_branch_p`, `k_path_p`, `k_ph_p`, `k_poly_p`, `k_t4p_p` and for each logic L in the modal cube transform each formula in a class so that is L-unsatisfiable, but L'-satisfiable for any logic L' that is not an extension of L. The transformation proceeds by first converting a formula φ to simplified NNF. Then for each propositional literal l it replaces all its occurrences by $(l \vee \psi_L^p)$ where $|l| = p$ and ψ_L^p is a modal formula uniquely associated with p and L, resulting in a formula φ'. Finally, for logics KD4 and KDB we need to add a disjunct $(\Box q \wedge \Box\neg q)$ to φ', while for logics S4 and KTB we need to add a disjunct $(q \wedge \Box\neg q)$, where q is a propositional symbol not occurring in φ'. These disjuncts are unsatisfiable in the respective logics but satisfiable in logics where D, or T, do not hold. Table 7 shows the formulae ψ_L^p that we use in our evaluation. In the table, q_p and q'_p are propositional variables uniquely associated with p that do not occur in φ. The overall effect of this transformation is that the resulting classes of formulae satisfy Principles (iii) and (v).

For satisfiable formulae, we use the five classes `k_poly_n`, `s4_md_n`, `s4_ph_n`, `s4_path_n`, `s4_s5_n` without modification. Although the first of these classes was designed to be K-satisfiable and the other four to be S4-satisfiable, the formulae in those classes are satisfiable in all the logics we consider. `s4_ipc_n` also consists

Table 8. Benchmarking results

Logic	Status	Total	GMR (cneg)	GMR (cord)	GMR (cplain)	R+MLR (cneg)	R+MLR (cord)	R+MLR (cplain)	LEO-III+E
K	S	100	84	85	77	**100**	**100**	**100**	0
KD	S	100	84	85	77	96	**100**	93	0
KT	S	100	70	**81**	50	66	68	61	0
KB	S	100	58	58	29	51	**64**	51	0
K4	S	100	83	**85**	77	56	57	50	0
K5	S	100	**67**	60	45	36	37	26	0
KDB	S	100	63	70	40	56	**73**	55	0
KTB	S	100	58	**59**	38	52	57	31	0
KD4	S	100	83	**85**	77	52	53	46	0
KD5	S	100	**73**	70	61	46	47	38	0
K45	S	100	45	**53**	34	36	37	25	0
K4B	S	100	18	19	11	23	**38**	15	0
KD45	S	100	**67**	66	56	46	47	38	0
S4	S	100	66	**76**	48	45	44	33	0
S5	S	100	32	28	32	32	**35**	24	0
All	S	1500	951	**980**	752	793	857	686	0
K	U	100	74	76	71	**79**	78	77	21
KD	U	100	74	**76**	71	73	75	62	13
KT	U	100	74	**77**	70	71	74	67	30
KB	U	100	71	**78**	68	71	52	55	10
K4	U	100	55	52	**57**	41	29	35	4
K5	U	100	74	46	**75**	50	30	48	8
KDB	U	100	73	**77**	71	73	52	56	8
KTB	U	100	72	**77**	69	67	50	53	9
KD4	U	100	**70**	59	67	40	32	39	1
KD5	U	100	75	46	**77**	51	40	46	3
K45	U	100	**51**	37	49	16	12	8	3
K4B	U	100	47	52	46	**53**	30	49	5
KD45	U	100	**64**	43	55	33	22	28	1
S4	U	100	47	**68**	66	45	21	23	4
S5	U	100	47	51	**52**	36	13	29	2
All	U	1500	**968**	915	964	799	610	675	122

only of S5-satisfiable formulae but these appear to be insufficiently challenging and have not been included in our benchmark set. All other classes of the LWB benchmark classes for K and S4 are satisfiable in some of the logics, but not in all. The five classes satisfy Principles (iv) and (v). The benchmark collection consisting of all ten classes together then also satisfies Principles (i) and (ii).

Another challenge for an empirical evaluation is the lack of available fully automatic theorem provers for all 15 logics that we have already discussed in Sect. 1. This leaves us with just three different approaches we can compare (i) the higher-order logic prover LEO-III [12,37], with **E** 2.6 as external reasoner, *LEO-III+E for short*, that supports a wide range of logics via semantic embedding into higher-order logic (ii) the combination of our reductions with the modal-layered resolution (MLR) calculus for SNF_{ml} clauses [25], *R+MLR calculus for short*, implemented in the modal theorem prover K$_\mathsf{S}$P (iii) the global modal resolution (GMR) calculus, implemented in K$_\mathsf{S}$P, which has resolution rules for all 15 logics [24]. For R+MLR and GMR calculi, resolution inferences between literal clauses can either be unrestricted (`cplain` option), restricted by negative resolution (`cneg` option), or restricted by an ordering (`cord` option). It is worth pointing out that negative and ordered resolution require slightly different transformations to the normal form that introduce additional clauses (`snf+` and `snf++` options, respectively). Also, the ordering cannot be arbitrary [25]. For the experiments, we have used the following options: (i) input processing: prenexing, together with simplification and pure literal elimination (`bnfsimp`, `prenex`, `early_ple`); (ii) preprocessing of clauses: renaming reuses symbols (`limited_reuse_renaming`), forward and backward subsumption (`fsub`, `bsub`) are enabled; the usable is populated with clauses whose maximal literal is positive (`populate_usable, max_lit_positive`); pure literal elimination is set for GMR (`ple`) and modal level ple is set for MLR (`mlple`); (iii) processing: inference rules not required for completeness are also used (`unit, lhs_unit,mres`), the options for preprocessing of clauses are kept and clause selection takes the shortest clause by level (`shortest`).

For LEO-III we provide the prover with a modal formula in the syntax it expects plus a logic specification that tells the prover in which modal logic the formula is meant to be solved, for example, `$modal_system_S4`. LEO-III can collaborate with external reasoners during proof search and we have used **E** 2.6 [34,35] as external reasoner and restricted LEO-III to one instance of **E** running in parallel. LEO-III is implemented in Java and we have set the maximum heap size to 1 GB and the thread stack size to 64 MB for the JVM.

Table 8 shows our benchmarking results. The first three columns of the table show the logic in which we determine the satisfiability status of each formula, the satisfiability status of the formulae, and their number. The next six columns then show how many of those formulae were solved by K$_\mathsf{S}$P with a particular calculus and refinement. The last column shows the result for LEO-III. The highest number or numbers are highlighted in bold. A time limit of 100 CPU seconds was set for each formula. Benchmarking was performed on a PC with an AMD Ryzen 5 5600X CPU @ 4.60 GHz max and 64 GB main memory using Fedora release 34 as operating system.

While the R+MLR calculus is competitive with GMR on extensions of K with axioms D, T and, possibly, B, the GMR calculus has better performance on extensions with axioms $\mathsf{4}$ and $\mathsf{5}$.

On satisfiable formulae, where for all logics we use exactly the same formulae and both resolution calculi have to saturate the set of clauses up to redundancy, the number of formulae solved is directly linked to the number of inferences necessary to do so. The fact that we reduce SNF_{sml} clauses to SNF_{ml} clauses via the introduction of multiple copies of the same clausal formulae with different labels clearly leads to a corresponding multiplication of the inferences that need to be performed. LEO-III+E does not solve any of the satisfiable formulae. This can be seen as an illustration of how important the use of additional techniques is that can turn resolution into a decision procedure on embeddings of modal logics into first-order logic [18,33].

On unsatisfiable formulae, where we use different formulae for each logic, the number of formulae solved is linked to the number of inferences it takes to find a refutation. For instance, on K it takes the GMR calculus on average 6.2 times the number of inferences to find a refutation than the R+MLR calculus. However, for all other logics the opposite is true. On the remaining 14 logics, the R+MLR calculus on average requires 6.5 times the number of inferences to find a refutation than the GMR calculus. Given that the R+MLR calculus currently uses a reduction from a modal logic to SNF_{sml} followed by a transformation from SNF_{sml} to SNF_{ml}, it is difficult to discern which of the two is the major problem. It is clear that multiple copies of the same clausal formulae are also detrimental to proof search. LEO-III+E does reasonably well on unsatisfiable formulae and the results clearly show the impact that additional axioms have on its performance. It performs best for KT and K but for logics involving axioms $\mathsf{4}$ and $\mathsf{5}$ very few formulae can be solved. The external prover **E** finds the proof for 121 out of the 122 modal formulae LEO-III+E can solve.

6 Conclusions

We have presented novel reductions of extensions of the modal logic K with arbitrary combinations of the axioms B, D, T, $\mathsf{4}$, $\mathsf{5}$ to clausal normal forms SNF_{sml} and SNF_{ml} for K. The implementation of those reductions combined with K_SP [26], allows us to reason in all 15 logics of the modal cube in a fully automatic way. Such support was so far extremely limited.

The transformation of sets of SNF_{sml} to sets of SNF_{ml} relies on new results that show that non-clausal closed tableaux in the Single Step Tableaux calculus [14,21] can be simulated by refutations in the modal-layered resolution (MLR) calculus for SNF_{ml} clauses [25].

We have also developed a new collection of benchmark formulae that covers all 15 logics of the modal cube. The collection consists of classes of parameterised and therefore scalable formulae. It contains an equal number of satisfiable and unsatisfiable formulae for each logic and the satisfiability status of each formula is known in advance. So far extensive collections of benchmark formulae were only available for K with smaller collections available for KT and $\mathsf{S4}$. A key feature of the approach is that it uses the systematic modification of K-unsatisfiable formulae to obtain unsatisfiable formulae in other logics. Thus, we could obtain a more extensive collection by applying this approach to further collections of benchmark formulae for K.

The evaluation we presented shows that on most of the 15 modal logics the combination of our reduction to SNF_{ml} with the MLR calculus does not perform as well as the global modal resolution (GMR) calculus, also implemented in K$_\mathrm{S}$P. This contrasts with the evaluation in [29], where we only considered six logics and used a different collection of benchmarks. We believe that the new benchmark collection more clearly indicates weaknesses in the current approach, in particular, the reduction from SNF_{sml} to SNF_{ml}. It is possible that the implementation of a calculus that operates directly on sets of SNF_{sml} clauses would perform considerably better as it avoids the repetition of clauses with different labels. However, it does so by using potentially infinite sets of labels which makes an implementation challenging. We intend to explore this possibility in future work.

References

1. Areces, C., de Rijke, M., de Nivelle, H.: Resolution in modal, description and hybrid logic. J. Log. Comput. **11**(5), 717–736 (2001)
2. Balsiger, P., Heuerding, A., Schwendimann, S.: A benchmark method for the propositional modal logics K, KT, S4. J. Autom. Reasoning **24**(3), 297–317 (2000). https://doi.org/10.1023/A:1006249507577
3. Basin, D., Matthews, S., Vigano, L.: Labelled propositional modal logics: theory and practice. J. Log. Comput. **7**(6), 685–717 (1997)
4. Blackburn, P., van Benthem, J., Wolter, F. (eds.): Handbook of Modal Logic. Elsevier (2006)
5. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (2002)
6. Brünnler, K.: Deep sequent systems for modal logic. Arch. Math. Log. **48**(6), 551–577 (2009). https://doi.org/10.1007/s00153-009-0137-3
7. Catach, L.: TABLEAUX: a general theorem prover for modal logics. J. Autom. Reason. **7**(4), 489–510 (1991). https://doi.org/10.1007/BF01880326
8. Chellas, B.F.: Modal Logic: An Introduction. Cambridge University Press, Cambridge (1980)
9. Fitting, M.: Prefixed tableaus and nested sequents. Ann. Pure Appl. Log. **163**(3), 291–313 (2012)
10. Gasquet, O., Herzig, A., Longin, D., Sahade, M.: LoTREC: logical tableaux research engineering companion. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS (LNAI), vol. 3702, pp. 318–322. Springer, Heidelberg (2005). https://doi.org/10.1007/11554554_25
11. Girlando, M., Straßburger, L.: MOIN: a nested sequent theorem prover for intuitionistic modal logics (system description). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12167, pp. 398–407. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51054-1_25
12. Gleißner, T., Steen, A.: LEO-III (2022). https://github.com/leoprover/Leo-III
13. Gleißner, T., Steen, A., Benzmüller, C.: Theorem provers for every normal modal logic. In: Eiter, T., Sands, D. (eds.) LPAR 2017. EPiC Series in Computing, vol. 46, pp. 14–30. EasyChair (2017). https://doi.org/10.29007/jsb9
14. Goré, R.: Tableau methods for modal and temporal logics. In: D'Agostino, M., Gabbay, D., Hähnle, R., Posegga, J. (eds.) Handbook of Tableau Methods, pp.

297–396. Springer, Heidelberg (1999). https://doi.org/10.1007/978-94-017-1754-0_6

15. Goré, R., Kikkert, C.: CEGAR-Tableaux: improved modal satisfiability via modal clause-learning and SAT. In: Das, A., Negri, S. (eds.) TABLEAUX 2021. LNCS (LNAI), vol. 12842, pp. 74–91. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86059-2_5
16. Governatori, G.: Labelled modal tableaux. In: Areces, C., Goldblatt, R. (eds.) AiML 2008, pp. 87–110. College Publications (2008)
17. Halpern, J.Y., Moses, Y.: A guide to completeness and complexity for modal logics of knowledge and belief. Artif. Intell. **54**(3), 319–379 (1992)
18. Horrocks, I., Hustadt, U., Sattler, U., Schmidt, R.A.: Computational modal logic. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic, chap. 4, pp. 181–245. Elsevier (2006)
19. Hustadt, U., de Nivelle, H., Schmidt, R.A.: Resolution-based methods for modal logics. Log. J. IGPL **8**(3), 265–292 (2000)
20. van Linder, B., van der Hoek, W., Meyer, J.J.C.: Formalising abilities and opportunities of agents. Fundamenta Informaticae **34**(1–2), 53–101 (1998)
21. Massacci, F.: Single step tableaux for modal logics. J. Autom. Reason. **24**, 319–364 (2000). https://doi.org/10.1023/A:1006155811656
22. Mayer, M.C.: Herbrand style proof procedures for modal logics. J. Appl. Non Class. Logics **3**(2), 205–223 (1993)
23. Nalon, C.: K_SP (2022). https://www.nalon.org/#software
24. Nalon, C., Dixon, C.: Clausal resolution for normal modal logics. J. Algorithms **62**, 117–134 (2007)
25. Nalon, C., Dixon, C., Hustadt, U.: Modal resolution: proofs, layers, and refinements. ACM Trans. Comput. Log. **20**(4), 23:1–23:38 (2019)
26. Nalon, C., Hustadt, U., Dixon, C.: K_SP: architecture, refinements, strategies and experiments. J. Autom. Reason. **64**(3), 461–484 (2020). https://doi.org/10.1007/s10817-018-09503-x
27. Ohlbach, H.J., Nonnengart, A., de Rijke, M., Gabbay, D.M.: Encoding two-valued nonclassical logics in classical logic. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, chap. 21, pp. 1403–1485. Elsevier (2001)
28. Otten, J.: MleanCoP: a connection prover for first-order modal logic. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 269–276. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_20
29. Papacchini, F., Nalon, C., Hustadt, U., Dixon, C.: Efficient local reductions to basic modal logic. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 76–92. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79876-5_5
30. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. J. Symb. Comput. **2**(3), 293–304 (1986)
31. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In: KR 1991, pp. 473–484. Morgan Kaufmann (1991)
32. Raths, T., Otten, J.: The QMLTP problem library for first-order modal logics. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 454–461. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_35
33. Schmidt, R.A., Hustadt, U.: First-order resolution methods for modal logics. In: Voronkov, A., Weidenbach, C. (eds.) Programming Logics. LNCS, vol. 7797, pp. 345–391. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37651-1_15

34. Schulz, S.: E 2.6 (2022). http://wwwlehre.dhbw-stuttgart.de/~sschulz/E/Download.html
35. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 495–507. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_29
36. Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: Giacomo, G.D., et al. (eds.) ECAI 2020. Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 2937–2938. IOS Press (2020). https://doi.org/10.3233/FAIA200462
37. Steen, A., Benzmüller, C.: Extensional higher-order paramodulation in Leo-III. J. Autom. Reason. **65**(6), 775–807 (2021). https://doi.org/10.1007/s10817-021-09588-x
38. The SPASS Team: SPASS 3.9 (2016). http://www.spass-prover.org/
39. Boy de la Tour, T.: An optimality result for clause form translation. J. Symb. Comput. **14**(4), 283–301 (1992)
40. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Siekmann, J.H., Wrightson, G. (eds.) Automation of Reasoning: Classical Papers on Computational Logic 1967–1970, vol. 2, pp. 466–483. Springer, Heidelberg (1983). https://doi.org/10.1007/978-3-642-81955-1_28. Original paper (in Russian) appeared in 1968
41. Wansing, H.: Sequent calculi for normal modal proposisional logics. J. Log. Comput. **4**(2), 125–142 (1994)
42. Weidenbach, C.: Combining superposition, sorts and splitting. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 1965–2013. Elsevier and MIT Press (2001)
43. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 140–145. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_10

Proof Systems and Proof Search

Cyclic Proofs, Hypersequents, and Transitive Closure Logic

Anupam Das(✉) and Marianna Girlando

University of Birmingham, Birmingham, UK
{a.das,m.girlando}@bham.ac.uk

Abstract. We propose a cut-free cyclic system for Transitive Closure Logic (TCL) based on a form of *hypersequents*, suitable for automated reasoning via proof search. We show that previously proposed sequent systems are cut-free incomplete for basic validities from Kleene Algebra (KA) and Propositional Dynamic Logic (PDL), over standard translations. On the other hand, our system faithfully simulates known cyclic systems for KA and PDL, thereby inheriting their completeness results. A peculiarity of our system is its richer correctness criterion, exhibiting 'alternating traces' and necessitating a more intricate soundness argument than for traditional cyclic proofs.

Keywords: Cyclic proofs · Transitive Closure Logic · Hypersequents · Propositional Dynamic Logic

1 Introduction

Transitive Closure Logic (TCL) is the extension of first-order logic by an operator computing the transitive closure of definable binary relations. It has been studied by numerous authors, e.g. [15–17], and in particular has been proposed as a foundation for the mechanisation and automation of mathematics [1].

Recently, Cohen and Rowe have proposed *non-wellfounded* and *cyclic* systems for TCL [9,11]. These systems differ from usual ones by allowing proofs to be infinite (finitely branching) trees, rather than finite ones, under some appropriate global correctness condition (the 'progressing criterion'). One particular feature of the cyclic approach to proof theory is the facilitation of automation, since complexity of inductive invariants is effectively traded off for a richer proof structure. In fact this trade off has recently been made formal, cf. [3,12], and has led to successful applications to automated reasoning, e.g. [6,7,24,26,27].

In this work we investigate the capacity of cyclic systems to automate reasoning in TCL. Our starting point is the demonstration of a key shortfall of Cohen and Rowe's system: its cut-free fragment, here called TC_G, is unable to cyclically prove even standard theorems of relational algebra, e.g. $(a \cup b)^* = a^*(ba^*)^*$ and

This work was supported by a UKRI Future Leaders Fellowship, 'Structure vs Invariants in Proofs', project reference MR/S035540/1.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 509–528, 2022.
https://doi.org/10.1007/978-3-031-10769-6_30

$(aa \cup aba)^+ \leq a^+((ba^+)^+ \cup a))$ (Theorem 12). An immediate consequence of this is that cyclic proofs of TC_G do not enjoy cut-admissibility (Corollary 13). On the other hand, these (in)equations are theorems of Kleene Algebra (KA) [18,19], a decidable theory which admits automation-via-proof-search thanks to the recent cyclic system of Das and Pous [14].

What is more, TCL is well-known to interpret Propositional Dynamic Logic (PDL), a modal logic whose modalities are just terms of KA, by a natural extension of the 'standard translation' from (multi)modal logic to first-order logic (see, e.g., [4,5]). Incompleteness of cyclic-TC_G for PDL over this translation is inherited from its incompleteness for KA. This is in stark contrast to the situation for modal logics without fixed points: the standard translation from K (and, indeed, all logics in the 'modal cube') to first-order logic actually *lifts* to cut-free proofs for a wide range of modal logic systems, cf. [21,22].

A closer inspection of the systems for KA and PDL reveals the stumbling block to any simulation: these systems implicitly conduct a form of 'deep inference', by essentially reasoning underneath $\exists$ and $\wedge$. Inspired by this observation, we propose a form of *hypersequents* for predicate logic, with extra structure admitting the deep reasoning required. We present the cut-free system HTC and a novel notion of cyclic proof for these hypersequents. In particular, the incorporation of some deep inference at the level of the rules necessitates an 'alternating' trace condition corresponding to *alternation* in automata theory.

Our first main result is the Soundness Theorem (Theorem 23): non-wellfounded proofs of HTC are sound for *standard semantics*. The proof is rather more involved than usual soundness arguments in cyclic proof theory, due to the richer structure of hypersequents and the corresponding progress criterion. Our second main result is the Simulation Theorem (Theorem 28): HTC is complete for PDL over the standard translation, by simulating a cut-free cyclic system for the latter. This result can be seen as a formal interpretation of cyclic modal proof theory within cyclic predicate proof theory, in the spirit of [21,22].

To simplify the exposition, we shall mostly focus on equality-free TCL and 'identity-free' PDL in this paper, though all our results hold also for the 'reflexive' extensions of both logics. We discuss these extensions in Sect. 7, and present further insights and conclusions in Sect. 8. Full proofs and further examples not included here (due to space constraints) can be found in [13].

2 Preliminaries

We shall work with a fixed first-order vocabulary consisting of a countable set Pr of unary *predicate* symbols, written p, q, etc., and of a countable set Rel of binary *relation* symbols, written a, b, etc. We shall generally reserve the word 'predicate' for unary and 'relation' for binary. We could include further relational symbols too, of higher arity, but choose not to in order to calibrate the semantics of both our modal and predicate settings.

We build formulas from this language differently in the modal and predicate settings, but all our formulas may be formally evaluated within *structures*:

Definition 1 (Structures). *A structure $\mathcal{M}$ consists of a set D, called the* domain *of $\mathcal{M}$, which we sometimes denote by $|\mathcal{M}|$; a subset $p^{\mathcal{M}} \subseteq D$ for each $p \in \mathsf{Pr}$; and a subset $a^{\mathcal{M}} \subseteq D \times D$ for each $a \in \mathsf{Rel}$.*

2.1 Transitive Closure Logic

In addition to the language introduced at the beginning of this section, in the predicate setting we further make use of a countable set of *function* symbols, written f^i, g^j, etc. where the superscripts $i, j \in \mathbb{N}$ indicate the *arity* of the function symbol and may be omitted when it is not ambiguous. Nullary function symbols (aka *constant* symbols), are written c, d etc. We shall also make use of *variables*, written x, y, etc., typically bound by quantifiers. *Terms*, written s, t, etc., are generated as usual from variables and function symbols by function application. A term is *closed* if it has no variables.

We consider the usual syntax for first-order logic formulas over our language, with an additional operator for transitive closure (and its dual). Formally, TCL formulas, written A, B, etc., are generated as follows:

$$\begin{aligned} A, B ::= {} & p(t) \mid \bar{p}(t) \mid a(s,t) \mid \bar{a}(s,t) \mid (A \wedge B) \mid (A \vee B) \mid \forall x A \mid \exists x A \mid \\ & TC(\lambda x, y.A)(s,t) \mid \overline{TC}(\lambda x, y.A)(s,t) \end{aligned}$$

When variables x, y are clear from context, we may write $TC(A(x,y))(s,t)$ or $TC(A)(s,t)$ instead of $TC(\lambda x, y.A)(s,t)$, as an abuse of notation, and similarly for $\overline{TC}$. We may write $A[t/x]$ for the formula obtained from A by replacing every free occurrence of the variable x by the term t. We have included both TC and $\overline{TC}$ as primitive operators, so that we can reduce negation to atomic formulas, shown below. This will eventually allow a one-sided formulation of proofs.

Definition 2 (Duality). *For a formula A we define its* complement*, $\bar{A}$, by:*

$$\begin{array}{rlrlrlrl} \overline{p(t)} & := \bar{p}(t) & \overline{\bar{p}(t)} & := p(t) & \overline{A \wedge B} & := \bar{A} \vee \bar{B} & \overline{TC(A)(s,t)} & := \overline{TC}(\bar{A})(s,t) \\ \overline{a(s,t)} & := \bar{a}(s,t) & \overline{\forall x A} & := \exists x \bar{A} & \overline{A \vee B} & := \bar{A} \wedge \bar{B} & \overline{\overline{TC}(A)(s,t)} & := TC(\bar{A})(s,t) \\ \overline{\bar{a}(s,t)} & := a(s,t) & \overline{\exists x A} & := \forall x \bar{A} & & & & \end{array}$$

We shall employ standard logical abbreviations, e.g. $A \supset B$ for $\bar{A} \vee B$.

We may evaluate formulas with respect to a structure, but we need additional data for interpreting function symbols:

Definition 3 (Interpreting function symbols). *Let $\mathcal{M}$ be a structure with domain D. An* interpretation *is a map ρ that assigns to each function symbol f^n a function $D^n \to D$. We may extend any interpretation ρ to an action on (closed) terms by setting recursively $\rho(f(t_1, \ldots, t_n)) := \rho(f)(\rho(t_1), \ldots, \rho(t_n))$.*

We only consider *standard* semantics in this work: TC (and $\overline{TC}$) is always interpreted as the *real* transitive closure (and its dual) in a structure, rather than being axiomatised by some induction (and coinduction) principle.

Definition 4 (Semantics). *Given a structure $\mathcal{M}$ with domain D and an interpretation ρ, the judgement $\mathcal{M}, \rho \models A$ is defined as usual for first-order logic with the following additional clauses for TC and $\overline{TC}$:*[1]

- *$\mathcal{M}, \rho \models TC(A(x,y))(s,t)$ if there are $v_0, \dots, v_{n+1} \in D$ with $\rho(s) = v_0$, $\rho(t) = v_{n+1}$, such that for every $i \leq n$ we have $\mathcal{M}, \rho \models A(v_i, v_{i+1})$.*
- *$\mathcal{M}, \rho \models \overline{TC}(A(x,y))(s,t)$ if for all $v_0, \dots, v_{n+1} \in D$ with $\rho(s) = v_0$ and $\rho(t) = v_{n+1}$, there is some $i \leq n$ such that $\mathcal{M}, \rho \models A(v_i, v_{i+1})$.*

If $\mathcal{M}, \rho \models A$ for all $\mathcal{M}$ and ρ, we simply write $\models A$.

Remark 5 (TC and $\overline{TC}$ as least and greatest fixed points). As expected, we have $\mathcal{M}, \rho \not\models TC(A)(s,t)$ just if $\mathcal{M}, \rho \models \overline{TC}(\bar{A})(s,t)$, and so the two operators are semantically dual. Thus, TC and $\overline{TC}$ duly correspond to least and greatest fixed points, respectively, satisfying in any model:

$$TC(A)(s,t) \iff A(s,t) \vee \exists x(A(s,x) \wedge TC(A)(x,t)) \quad (1)$$
$$\overline{TC}(A)(s,t) \iff A(s,t) \wedge \forall x(A(s,x) \vee \overline{TC}(A)(x,t)) \quad (2)$$

Let us point out that our $\overline{TC}$ operator is not the same as Cohen and Rowe's transitive 'co-closure' operator TC^{op} in [10], but rather the De Morgan dual of TC. In the presence of negation, TC and $\overline{TC}$ are indeed interdefinable, cf. Definition 2.

2.2 Cohen-Rowe Cyclic System for TCL

Cohen and Rowe proposed in [9,11] a non-wellfounded system for TCL that extends a usual sequent calculus $\mathsf{LK}_=$ for first-order logic with equality and substitution by rules for TC inspired by its characterisation as a least fixed point, cf. (1).[2] Note that the presence of the substitution rule is critical for the notion of 'regularity' in predicate cyclic proof theory. The resulting notions of non-wellfounded and cyclic proofs are formulated similarly to those for first-order logic with (ordinary) inductive definitions [8]:

Definition 6 (Sequent system). TC_G *is the extension of* $\mathsf{LK}_=$ *by the rules:*

$$\frac{\Gamma, A(s,t)}{\Gamma, TC(A)(s,t)}\, {\scriptstyle TC_0} \qquad \frac{\Gamma, A(s,r) \quad \Gamma, TC(A)(r,t)}{\Gamma, TC(A)(s,t)}\, {\scriptstyle TC_1}$$
$$\frac{\Gamma, A(s,t) \quad \Gamma, A(s,c), \overline{TC}(A)(c,t)}{\Gamma, \overline{TC}(A)(s,t)}\, {\scriptstyle \overline{TC}} \ c \text{ fresh} \quad (3)$$

TC_G-preproofs *are possibly infinite trees of sequents generated by the rules of* TC_G. *A preproof is* regular *if it has only finitely many distinct sub-preproofs.*

[1] Note that we are including 'parameters from the model' in formulas here. Formally, this means each $v \in D$ is construed as a constant symbol for which $\rho(v) = v$.

[2] Cohen and Rowe's system is originally called RTC_G, rather using a 'reflexive' version RTC of the TC operator. However this (and its rules) can be encoded (and simulated) by defining $RTC(\lambda x, y.A)(s,t) := TC(\lambda x, y(x = y \vee A))(s,t)$.

The notion of 'correct' non-wellfounded proof is obtained by a standard *progressing criterion* in cyclic proof theory. We shall not go into details here, being beyond the scope of this work, but refer the reader to those original works (as well as [13] for our current variant). Let us write $\vdash_{cyc}$ for their notion of cyclic provability using the above rules, cf. [9,11]. A standard infinite descent countermodel argument yields:

Proposition 7 (Soundness, [9,11]). *If* $\mathsf{TC}_G \vdash_{cyc} A$ *then* $\models A$.

In fact, this result is subsumed by our main soundness result for HTC (Theorem 23) and its simulation of TC_G (Theorem 19). In the presence of cut, a form of converse of Proposition 7 holds: cyclic TC_G proofs are 'Henkin complete', i.e. complete for all models of a particular axiomatisation of TCL based on (co)induction principles for TC (and $\overline{TC}$) [9,11]. However, the counterexample we present in the next section implies that cut is not eliminable (Corollary 13).

3 Interlude: Motivation from PDL and Kleene Algebra

Given the TCL sequent system proposed by Cohen and Rowe, why do we propose a hypersequential system? Our main argument is that proof search in TC_G is rather weak, to the extent that cut-free cyclic proofs are unable to simulate a basic (cut-free) system for modal logic PDL (regardless of proof search strategy). At least one motivation here is to 'lift' the *standard translation* from cut-free cyclic proofs for PDL to cut-free cyclic proofs in an adequate system for TCL.

3.1 Identity-Free PDL

Identity-free propositional dynamic logic (PDL$^+$) is a version of the modal logic PDL without tests or identity, thereby admitting an 'equality-free' standard translation into predicate logic. Formally, PDL$^+$ *formulas*, written A, B, etc., and *programs*, written α, β, etc., are generated by the following grammars:

$$\begin{aligned} A, B &::= p \mid \overline{p} \mid (A \wedge B) \mid (A \vee B) \mid [\alpha]A \mid \langle\alpha\rangle A \\ \alpha, \beta &::= a \mid (\alpha;\beta) \mid (\alpha \cup \beta) \mid \alpha^+ \end{aligned}$$

We sometimes simply write $\alpha\beta$ instead of $\alpha;\beta$, and $(\alpha)A$ for a formula that is either $\langle\alpha\rangle A$ or $[\alpha]A$.

Definition 8 (Duality). *For a formula A we define its* complement*, $\bar{A}$, by:*

$$\bar{\bar{p}} := p \qquad \begin{aligned} \overline{A \wedge B} &:= \bar{A} \vee \bar{B} \\ \overline{A \vee B} &:= \bar{A} \wedge \bar{B} \end{aligned} \qquad \begin{aligned} \overline{[\alpha]A} &:= \langle\alpha\rangle\bar{A} \\ \overline{\langle\alpha\rangle A} &:= [\alpha]\bar{A} \end{aligned}$$

We *evaluate* PDL$^+$ formulas using the traditional relational semantics of modal logic, by associating each program with a binary relation in a structure. Again, we only consider standard semantics, in the sense that the $+$ operator is interpreted as the real transitive closure within a structure.

Definition 9 (Semantics). *For structures $\mathcal{M}$ with domain D, elements $v \in D$, programs α and formulas A, we define $\alpha^{\mathcal{M}} \subseteq D \times D$ and the judgement $\mathcal{M}, v \models A$ as follows:*

- *($a^{\mathcal{M}}$ is already given in the specification of $\mathcal{M}$, cf. Definition 1).*
- *$(\alpha;\beta)^{\mathcal{M}} := \{(u, v) :$ there is $w \in D$ s.t. $(u, w) \in \alpha^{\mathcal{M}}$ and $(w, v) \in \beta^{\mathcal{M}}\}$.*
- *$(\alpha \cup \beta)^{\mathcal{M}} := \{(u, v) : (u, v) \in \alpha^{\mathcal{M}}$ or $(u, v) \in \beta^{\mathcal{M}}\}$.*
- *$(\alpha^+)^{\mathcal{M}} := \{(u, v) :$ there are $w_0, \dots, w_{n+1} \in D$ s.t. $u = w_0, v = w_{n+1}$ and, for every $i \leq n, (w_i, w_{i+1}) \in \alpha^{\mathcal{M}}\}$.*

- *$\mathcal{M}, v \models p$ if $v \in p^{\mathcal{M}}$.*
- *$\mathcal{M}, v \models \overline{p}$ if $v \notin p^{\mathcal{M}}$.*
- *$\mathcal{M}, v \models A \wedge B$ if $\mathcal{M}, v \models A$ and $\mathcal{M}, v \models B$.*
- *$\mathcal{M}, v \models A \vee B$ if $\mathcal{M}, v \models A$ or $\mathcal{M}, v \models B$.*
- *$\mathcal{M}, v \models [\alpha]A$ if $\forall\,(v, w) \in \alpha^{\mathcal{M}}$ we have $\mathcal{M}, w \models A$.*
- *$\mathcal{M}, v \models \langle\alpha\rangle A$ if $\exists\,(v, w) \in \alpha^{\mathcal{M}}$ with $\mathcal{M}, w \models A$.*

If $\mathcal{M}, v \models A$ for all $\mathcal{M}$ and $v \in |\mathcal{M}|$, then we write $\models A$.

Note that we are overloading the satisfaction symbol $\models$ here, for both PDL$^+$ and TCL. This should never cause confusion, in particular since the two notions of satisfaction are 'compatible' as we shall now see.

3.2 The Standard Translation

The so-called 'standard translation' of modal logic into predicate logic is induced by reading the semantics of modal logic as first-order formulas. We now give a natural extension of this that interprets PDL$^+$ into TCL. At the logical level our translation coincides with the usual one for basic modal logic; our translation of programs, as expected, requires the TC operator to interpret the $+$ of PDL$^+$.

Definition 10. *For* PDL$^+$ *formulas A and programs α, we define the* standard translations $\mathsf{ST}(A)(x)$ *and* $\mathsf{ST}(\alpha)(x, y)$ *as* TCL*-formulas with free variables x and x, y, resp., inductively as follows:*

$$\begin{array}{rclrcl}
\mathsf{ST}(p)(x) &:=& p(x) & \mathsf{ST}(a)(x,y) &:=& a(x,y) \\
\mathsf{ST}(\bar{p})(x) &:=& \bar{p}(x) & \mathsf{ST}(\alpha\cup\beta)(x,y) &:=& \mathsf{ST}(\alpha)(x,y) \vee \mathsf{ST}(\beta)(x,y) \\
\mathsf{ST}(A \vee B)(x) &:=& \mathsf{ST}(A)(x) \vee \mathsf{ST}(B)(x) & \mathsf{ST}(\alpha;\beta)(x,y) &:=& \exists z(\mathsf{ST}(\alpha)(x,z) \wedge \mathsf{ST}(\beta)(z,y)) \\
\mathsf{ST}(A \wedge B)(x) &:=& \mathsf{ST}(A)(x) \wedge \mathsf{ST}(B)(x) & \mathsf{ST}(\alpha^+)(x,y) &:=& TC(\mathsf{ST}(\alpha))(x,y) \\
\mathsf{ST}(\langle\alpha\rangle A)(x) &:=& \exists y(\mathsf{ST}(\alpha)(x,y) \wedge \mathsf{ST}(A)(y)) & & & \\
\mathsf{ST}([\alpha]A)(x) &:=& \forall y(\overline{\mathsf{ST}(\alpha)(x,y)} \vee \mathsf{ST}(A)(y)) & & &
\end{array}$$

where $TC(\mathsf{ST}(\alpha))$ is shorthand for $TC(\lambda x, y.\mathsf{ST}(\alpha)(x, y))$.

It is routine to show that $\overline{\mathsf{ST}(A)(x)} = \mathsf{ST}(\bar{A})(x)$, by structural induction on A, justifying our overloading of the notation $\bar{A}$, in both TCL and PDL$^+$. Yet another advantage of using the same underlying language for both the modal and predicate settings is that we can state the following (expected) result without the need for encodings, following by a routine structural induction (see, e.g., [5]):

Theorem 11. *For* PDL$^+$ *formulas A, we have $\mathcal{M}, v \models A$ iff $\mathcal{M} \models \mathsf{ST}(A)(v)$.*

3.3 Cohen-Rowe System is not Complete for PDL$^+$

PDL$^+$ admits a standard cut-free cyclic proof system LPD^+ (see Sect. 6.1) which is both sound and complete (cf. Theorem 30). However, a shortfall of TC_G is that it is unable to cut-free simulate LPD^+. In fact, we can say something stronger:

Theorem 12 (Incompleteness). *There exist a* PDL$^+$ *formula* A *such that* $\models A$ *but* $\mathsf{TC}_G \not\vdash_{cyc} \mathsf{ST}(A)(x)$ *(in the absence of cut).*

This means not only that TC_G is unable to locally cut-free simulate the rules of LPD^+, but also that there are some validities for which there are no cut-free cyclic proofs at all in TC_G. One example of such a formula is:

$$\langle (aa \cup aba)^+ \rangle p \supset \langle a^+((ba^+)^+ \cup a)\rangle p \tag{4}$$

A detailed proof of this is found in [13], but let us briefly discuss it here. First, the formula above is not artificial: it is derived from the well-known PDL validity $\langle (a \cup b)^* \rangle p \supset \langle a^*(ba^*)^* \rangle p$ by identity-elimination. This in turn is essentially a theorem of relational algebra, namely $(a \cup b)^* \leq a^*(ba^*)^*$, which is often used to eliminate $\cup$ in (sums of) regular expressions. The same equation was (one of those) used by Das and Pous in [14] to show that the sequent system LKA for Kleene Algebra is cut-free cyclic incomplete.

The argument that $\mathsf{TC}_G \not\vdash_{cyc} \mathsf{ST}(4)(x)$ is much more involved than the one from [14], due to the fact we are working in predicate logic, but the underlying basic idea is similar. At a very high level, the RHS of (4) (viewed as a relational inequality) is translated to an existential formula $\exists z(\mathsf{ST}(a^+)(x,z) \wedge \mathsf{ST}((ba^+)^+ \cup a)(z,y)$ that, along some branch (namely the one that always chooses aa when decomposing the LHS of (4)) can never be instantiated while remaining valid. This branch witnesses the non-regularity of any proof. However $\mathsf{ST}(4)(x)$ is cyclically provable in TC_G with cut, so an immediate consequence of Theorem 12 is:

Corollary 13. *The class of cyclic proofs of* TC_G *does not enjoy cut-admissibility.*

4 Hypersequent Calculus for TCL

Let us take a moment to examine why any 'local' simulation of LPD^+ by TC_G fails, in order to motivate the main system that we shall present. The program rules, in particular the $\langle\,\rangle$-rules, require a form of *deep inference* to be correctly simulated, over the standard translation. For instance, let us consider the action of the standard translation on two rules we shall see later in LPD^+ (cf. Sect. 6.1):

$$\langle\cup\rangle_0\,\frac{\Gamma, \langle a_0\rangle p}{\Gamma, \langle a_0 \cup a_1\rangle p} \quad \rightsquigarrow \quad \frac{\mathsf{ST}(\Gamma)(c), \exists x(a_0(c,x) \wedge p(x))}{\mathsf{ST}(\Gamma)(c), \exists x((a_0(c,x) \vee a_1(c,x)) \wedge p(x))}$$

$$\langle;\rangle\,\frac{\Gamma, \langle a\rangle\langle b\rangle p}{\Gamma, \langle a;b\rangle p} \quad \rightsquigarrow \quad \frac{\mathsf{ST}(\Gamma)(c), \exists y(a(c,y) \wedge \exists x(b(y,x) \wedge p(x)))}{\mathsf{ST}(\Gamma)(c), \exists x(\exists y(a(c,y) \wedge b(y,x)) \wedge p(x))}$$

$$\mathsf{init}\ \frac{}{\{\}^{\varnothing}} \qquad \mathsf{wk}\ \frac{\mathbf{S}}{\mathbf{S},\mathbf{S}'} \qquad \sigma\ \frac{\mathbf{S}}{\sigma(\mathbf{S})} \qquad \cup\ \frac{\mathbf{S},\{\Gamma\}^{\mathbf{x}} \quad \mathbf{S},\{\Delta\}^{\mathbf{y}}}{\mathbf{S},\{\Gamma,\Delta\}^{\mathbf{x},\mathbf{y}}}\ \begin{array}{l}\mathsf{fv}(\Delta)\cap\mathbf{x}=\varnothing\\ \mathsf{fv}(\Gamma)\cap\mathbf{y}=\varnothing\end{array}$$

$$\mathsf{id}\ \frac{\mathbf{S},\{\Gamma\}^{\mathbf{x}}}{\mathbf{S},\{\Gamma,A\}^{\mathbf{x}},\{\overline{A}\}^{\varnothing}}\ A \text{ closed} \qquad \wedge\ \frac{\mathbf{S},\{\Gamma,A,B\}^{\mathbf{x}}}{\mathbf{S},\{\Gamma,A\wedge B\}^{\mathbf{x}}} \qquad \vee\ \frac{\mathbf{S},\{\Gamma,A_i\}^{\mathbf{x}}}{\mathbf{S},\{\Gamma,A_0\vee A_1\}^{\mathbf{x}}}\ i\in\{0,1\}$$

$$\mathsf{inst}\ \frac{\mathbf{S},\{\Gamma(t)\}^{\mathbf{x}}}{\mathbf{S},\{\Gamma(y)\}^{\mathbf{x},y}} \qquad \exists\ \frac{\mathbf{S},\{\Gamma,A(y)\}^{\mathbf{x},y}}{\mathbf{S},\{\Gamma,\exists x(A(x))\}^{\mathbf{x}}}\ y \text{ fresh} \qquad \forall\ \frac{\mathbf{S},\{\Gamma,A(f(\mathbf{x}))\}^{\mathbf{x}}}{\mathbf{S},\{\Gamma,\forall x(A(x))\}^{\mathbf{x}}}\ f \text{ fresh}$$

$$TC\ \frac{\mathbf{S},\{\Gamma,A(s,t)\}^{\mathbf{x}},\{\Gamma,A(s,z),TC(A)(z,t)\}^{\mathbf{x},z}}{\mathbf{S},\{\Gamma,TC(A)(s,t)\}^{\mathbf{x}}}\ z \text{ fresh}$$

$$\overline{TC}\ \frac{\mathbf{S},\{\Gamma,A(s,t),A(s,f(\mathbf{x}))\}^{\mathbf{x}},\{\Gamma,A(s,t),\overline{TC}(A)(f(\mathbf{x}),t)\}^{\mathbf{x}}}{\mathbf{S},\{\Gamma,\overline{TC}(A)(s,t)\}^{\mathbf{x}}}\ f \text{ fresh}$$

Fig. 1. Hypersequent calculus HTC. σ is a 'substitution' map from constants to terms and a renaming of other function symbols and variables.

The first case above suggests that any system to which the standard translation lifts must be able to reason *underneath* $\exists$ *and* $\wedge$, so that the inference indicated in blue is 'accessible' to the prover. The second case above suggests that the existential-conjunctive meta-structure necessitated by the first case should admit basic equivalences, in particular certain *prenexing*. This section is devoted to the incorporation of these ideas (and necessities) into a bona fide proof system.

4.1 A System for Predicate Logic via Annotated Hypersequents

An *annotated cedent*, or simply *cedent*, written S, S' etc., is an expression $\{\Gamma\}^{\mathbf{x}}$, where Γ is a set of formulas and the *annotation* $\mathbf{x}$ is a set of variables. We sometimes construe annotations as lists rather than sets when it is convenient, e.g. when taking them as inputs to a function.

Each cedent may be intuitively read as a TCL formula, under the following interpretation: $\mathit{fm}(\{\Gamma\}^{x_1,\dots,x_n}) := \exists x_1 \dots \exists x_n \bigwedge \Gamma$. When $\mathbf{x} = \varnothing$ then there are no existential quantifiers above, and when $\Gamma = \varnothing$ we simply identify $\bigwedge \Gamma$ with $\top$. We also sometimes write simply A for the annotated cedent $\{A\}^{\varnothing}$.

A *hypersequent*, written $\mathbf{S}, \mathbf{S}'$ etc., is a set of annotated cedents. Each hypersequent may be intuitively read as the disjunction of its cedents. Namely we set: $\mathit{fm}(\{\Gamma_1\}^{\mathbf{x}_1}, \dots, \{\Gamma_n\}^{\mathbf{x}_n}) := \mathit{fm}(\{\Gamma_1\}^{\mathbf{x}_1}) \vee \dots \vee \mathit{fm}(\{\Gamma_n\}^{\mathbf{x}_n})$.

Definition 14 (System). *The rules of* HTC *are given in Fig. 1. A* HTC pre-proof *is a (possibly infinite) derivation tree generated by the rules of* HTC. *A preproof is* regular *if it has only finitely many distinct subproofs.*

Our hypersequential system is somewhat more refined than usual sequent systems for predicate logic. E.g., the usual $\exists$ rule is decomposed into $\exists$ and inst, whereas the usual $\wedge$ rule is decomposed into $\wedge$ and $\cup$. The rules for TC and $\overline{TC}$ are induced directly from their characterisations as fixed points in (1).

Note that the rules $\overline{TC}$ and $\forall$ introduce, bottom-up, the fresh function symbol f, which plays the role of the *Herbrand function* of the corresponding $\forall$ quantifier: just as $\forall \mathbf{x} \exists x A(x)$ is equisatisfiable with $\forall \mathbf{x} A(f(\mathbf{x}))$, when f is fresh, by Skolemisation, by duality $\exists \mathbf{x} \forall x A(x)$ is equivalid with $\exists \mathbf{x} A(f(\mathbf{x}))$, when f is fresh, by Herbrandisation. The usual $\forall$ rule of the sequent calculus corresponds to the case when $\mathbf{x} = \varnothing$.

4.2 Non-wellfounded Hypersequent Proofs

Our notion of ancestry, as compared to traditional sequent systems, must account for the richer structure of hypersequents:

Definition 15 (Ancestry). *Fix an inference step* r*, as typeset in Fig. 1. A formula C in the premiss is an* immediate ancestor *of a formula C' in the conclusion if they have the same colour; if $C, C' \in \Gamma$ then we further require $C = C'$, and if C, C' occur in $\mathbf{S}$ then $C = C'$ occur in the same cedent. A cedent S in the premiss is an immediate ancestor of a cedent S' in the conclusion if some formula in S is an* immediate ancestor *of some formula in S'.*

Immediate ancestry on both formulas and cedents is a binary relation, inducing a directed graph whose paths form the basis of our correctness condition:

Definition 16 ((Hyper)traces). *A* hypertrace *is a maximal path in the graph of immediate ancestry on cedents. A* trace *is a maximal path in the graph of immediate ancestry on formulas.*

Definition 17 (Progress and proofs). *Fix a preproof $\mathcal{D}$. A (infinite) trace $(F_i)_{i\in\omega}$ is* progressing *if there is k such that, for all $i > k$, F_i has the form $\overline{TC}(A)(s_i, t_i)$ and is infinitely often principal.*[3] *A (infinite) hypertrace $\mathcal{H}$ is* progressing *if every infinite trace within it is progressing. A (infinite) branch is* progressing *if it has a progressing hypertrace. $\mathcal{D}$ is a* proof *if every infinite branch is progressing. If, furthermore, $\mathcal{D}$ is regular, we call it a* cyclic proof.

We write $\mathsf{HTC} \vdash_{nwf} \mathbf{S}$ *(or* $\mathsf{HTC} \vdash_{cyc} \mathbf{S}$*) if there is a proof (or cyclic proof, respectively) of* HTC *of the hypersequent* $\mathbf{S}$*.*

In usual cyclic systems, checking that a regular preproof is progressing is decidable by straightforward reduction to the universality of nondeterministic ω-automata, with runs 'guessing' a progressing trace along an infinite branch. Our notion of progress exhibits an extra quantifier alternation: we must *guess* an infinite hypertrace in which *every* trace is progressing. Nonetheless, by appealing to determinisation or alternation, we can still decide our progressing condition:

Proposition 18. *Checking whether a* HTC *preproof is a proof is decidable by reduction to universality of ω-regular languages.*

[3] In fact, by a simple well-foundedness argument, it is equivalent to say that $(F_i)_{i<\omega}$ is progressing if it is infinitely often principal for a $\overline{TC}$-formula.

As we mentioned earlier, cyclic proofs of HTC indeed are at least as expressive as those of Cohen and Rowe's system by a routine local simulation of rules:

Theorem 19 (Simulating Cohen-Rowe). *If* $\mathsf{TC}_G \vdash_{cyc} A$ *then* $\mathsf{HTC} \vdash_{cyc} A$.

4.3 Some Examples

Example 20 (Fixed point identity). The sequent $\{\overline{TC}(a)(c,d)\}^{\varnothing}, \{TC(\bar{a})(c,d)\}^{\varnothing}$ is finitely derivable using rule id on $TC(a)(c,d)$ and the init rule. However we can also cyclically reduce it to a simpler instance of id. Due to the granularity of the inference rules of HTC, we actually have some liberty in how we implement such a derivation. E.g., the HTC-proof below applies TC rules below $\overline{TC}$ ones, and delays branching until the 'end' of proof search, which is impossible in TC_G. The only infinite branch, looping on $\bullet$, is progressing by the blue hypertrace.

$$
\begin{prooftree}
\AxiomC{}
\LeftLabel{init}
\UnaryInfC{$\{\,\}^{\varnothing}$}
\LeftLabel{id}
\UnaryInfC{$\{a(c,d)\}^{\varnothing}, \{\bar{a}(c,d)\}^{\varnothing}$}
\AxiomC{}
\LeftLabel{init}
\UnaryInfC{$\{\,\}^{\varnothing}$}
\LeftLabel{id}
\UnaryInfC{$\{a(c,e)\}^{\varnothing}, \{\bar{a}(c,e)\}^{\varnothing}$}
\AxiomC{$\vdots$}
\LeftLabel{TC}
\RightLabel{$\bullet$}
\UnaryInfC{${\color{blue}\{\overline{TC}(a)(e,d)\}^{\varnothing}}, \{TC(\bar{a})(e,d)\}^{\varnothing}$}
\LeftLabel{$\cup$}
\BinaryInfC{$\{a(c,e)\}^{\varnothing}, {\color{blue}\{\overline{TC}(a)(e,d)\}^{\varnothing}}, \{\bar{a}(c,e), TC(\bar{a})(e,d)\}^{\varnothing}$}
\LeftLabel{$2\cup$}
\BinaryInfC{$\{a(c,d), a(c,e)\}^{\varnothing}, {\color{blue}\{a(c,d), \overline{TC}(a)(e,d)\}^{\varnothing}}, \{\bar{a}(c,d)\}^{\varnothing}, \{\bar{a}(c,e), TC(\bar{a})(e,d)\}^{\varnothing}$}
\LeftLabel{inst}
\UnaryInfC{$\{a(c,d), a(c,e)\}^{\varnothing}, {\color{blue}\{a(c,d), \overline{TC}(a)(e,d)\}^{\varnothing}}, \{\bar{a}(c,d)\}^{\varnothing}, \{\bar{a}(c,x), TC(\bar{a})(x,d)\}^{x}$}
\LeftLabel{$\overline{TC}$}
\UnaryInfC{${\color{blue}\{\overline{TC}(a)(c,d)\}^{\varnothing}}, \{\bar{a}(c,d)\}^{\varnothing}, \{\bar{a}(c,x), TC(\bar{a})(x,d)\}^{x}$}
\LeftLabel{TC}
\RightLabel{$\bullet$}
\UnaryInfC{${\color{blue}\{\overline{TC}(a)(c,d)\}^{\varnothing}}, \{TC(\bar{a})(c,d)\}^{\varnothing}$}
\end{prooftree}
$$

This is an example of the more general 'rule permutations' available in HTC, hinting at a more flexible proof theory (we discuss this further in Sect. 8).

Example 21 (Transitivity). TC can be proved transitive by way of a cyclic proof in TC_G of the sequent $\overline{TC}(a)(c,d), \overline{TC}(a)(d,e), TC(\bar{a})(c,e)$. As in the previous example we may mimic that proof line by line, but we give a slightly different one that cannot directly be interpreted as a TC_G proof:

$$
\begin{prooftree}
\AxiomC{}
\LeftLabel{init}
\UnaryInfC{$\{\,\}^{\varnothing}$}
\LeftLabel{id}
\UnaryInfC{$a(c,d), \bar{a}(c,d)$}
\AxiomC{Ex. 20}
\doubleLine
\UnaryInfC{$TC(\bar{a})(d,e), \overline{TC}(a)(d,e)$}
\LeftLabel{$\cup$}
\BinaryInfC{$a(c,d), \overline{TC}(a)(d,e), \bar{a}(c,d), TC(\bar{a})(d,e)$}
\LeftLabel{inst}
\UnaryInfC{$a(c,d), \overline{TC}(a)(d,e), \{\bar{a}(c,x), TC(\bar{a})(x,e)\}^{x}$}
\AxiomC{}
\LeftLabel{init}
\UnaryInfC{$\{\,\}^{\varnothing}$}
\LeftLabel{id}
\UnaryInfC{$a(c,c'), \bar{a}(c,c')$}
\AxiomC{$\vdots$}
\LeftLabel{TC}
\RightLabel{$\circ$}
\UnaryInfC{${\color{red}\overline{TC}(a)(c',d)}, \overline{TC}(a)(d,e), TC(\bar{a})(c',e)$}
\LeftLabel{$\cup$}
\BinaryInfC{$a(c,c'), {\color{red}\overline{TC}(a)(c',d)}, \overline{TC}(a)(d,e), \bar{a}(c,e), \bar{a}(c,c'), TC(\bar{a})(c',e)$}
\LeftLabel{inst}
\UnaryInfC{$a(c,c'), {\color{red}\overline{TC}(a)(c',d)}, \overline{TC}(a)(d,e), \bar{a}(c,e), \{\bar{a}(c,x), TC(\bar{a})(x,e)\}^{x}$}
\LeftLabel{$2\cup$}
\BinaryInfC{$a(c,d), a(c,c'), {\color{red}a(c,d), \overline{TC}(a)(c',d)}, \overline{TC}(a)(d,e), \bar{a}(c,e), \{\bar{a}(c,x), TC(\bar{a})(x,e)\}^{x}$}
\LeftLabel{$\overline{TC}$}
\UnaryInfC{${\color{red}\overline{TC}(a)(c,d)}, \overline{TC}(a)(d,e), \bar{a}(c,e), \{\bar{a}(c,x), TC(\bar{a})(x,e)\}^{x}$}
\LeftLabel{TC}
\RightLabel{$\circ$}
\UnaryInfC{${\color{red}\overline{TC}(a)(c,d)}, \overline{TC}(a)(d,e), TC(\bar{a})(c,e)$}
\end{prooftree}
$$

The only infinite branch (except for that from Example 20), looping on $\circ$, is progressing by the red hypertrace.

Finally, it is pertinent to revisit the 'counterexample' (4) that witnessed incompleteness of TC_G for PDL$^+$. The following result is, in fact, already implied by our later completeness result, Theorem 28, but we shall present it nonetheless:

Proposition 22. $\mathsf{HTC} \vdash_{cyc} \mathsf{ST}((aa \cup aba)^+)(c,d) \supset \mathsf{ST}(a^+((ba^+)^+ \cup a))(c,d)$.

Proof. We give the required cyclic proof in Fig. 2, using the abbreviations: $\alpha(c,d) = \mathsf{ST}(aa \cup aba)(c,d)$ and $\beta(c,d) = \mathsf{ST}((ba^+)^+ \cup a)(c,d)$. The only infinite branch (looping on •) has progressing hypertrace is marked in blue. Hypersequents $\mathbf{R} = \{\overline{\alpha}(c,d)\}^{\varnothing}, \{\overline{\alpha}(c,d), \overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{TC(a)(c,y), \beta(y,d)\}^{y}$ and $\mathbf{R}' = \{\overline{\alpha}(c,d)\}^{\varnothing}, \{\overline{\alpha}(c,d)\}^{\varnothing}, \{TC(a)(c,y), \beta(y,d)\}^{y}$ have finitary proofs, while $\mathbf{P} = \{\overline{aba}(c,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{TC(a)(c,y), \beta(y,d)\}^{y}$ has a cyclic proof.

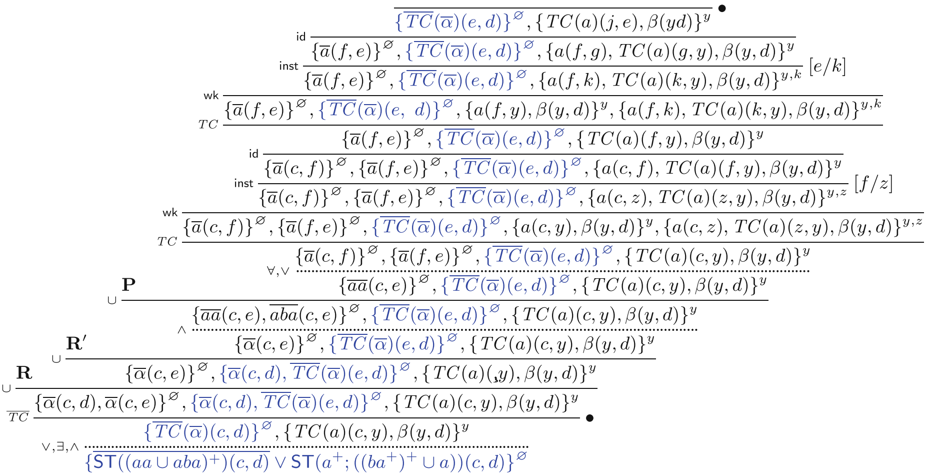

$\{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{TC(a)(j,e), \beta(yd)\}^{y}$ •
id
$\{\overline{a}(f,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{a(f,g), TC(a)(g,y), \beta(y,d)\}^{y}$
inst
$\{\overline{a}(f,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{a(f,k), TC(a)(k,y), \beta(y,d)\}^{y,k}$ $[e/k]$
wk
$\{\overline{a}(f,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,\ d)\}^{\varnothing}, \{a(f,y), \beta(y,d)\}^{y}, \{a(f,k), TC(a)(k,y), \beta(y,d)\}^{y,k}$
TC
$\{\overline{a}(f,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{TC(a)(f,y), \beta(y,d)\}^{y}$
id
$\{\overline{a}(c,f)\}^{\varnothing}, \{\overline{a}(f,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{a(c,f), TC(a)(f,y), \beta(y,d)\}^{y}$
inst
$\{\overline{a}(c,f)\}^{\varnothing}, \{\overline{a}(f,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{a(c,z), TC(a)(z,y), \beta(y,d)\}^{y,z}$ $[f/z]$
wk
$\{\overline{a}(c,f)\}^{\varnothing}, \{\overline{a}(f,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{a(c,y), \beta(y,d)\}^{y}, \{a(c,z), TC(a)(z,y), \beta(y,d)\}^{y,z}$
TC
$\{\overline{a}(c,f)\}^{\varnothing}, \{\overline{a}(f,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{TC(a)(c,y), \beta(y,d)\}^{y}$
$\forall,\vee$
$\mathbf{P}$
$\{\overline{aa}(c,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{TC(a)(c,y), \beta(y,d)\}^{y}$
$\cup$
$\{\overline{aa}(c,e), \overline{aba}(c,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{TC(a)(c,y), \beta(y,d)\}^{y}$
$\wedge$
$\mathbf{R}'$
$\{\overline{\alpha}(c,e)\}^{\varnothing}, \{\overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{TC(a)(c,y), \beta(y,d)\}^{y}$
$\cup$
$\mathbf{R}$
$\{\overline{\alpha}(c,e)\}^{\varnothing}, \{\overline{\alpha}(c,d), \overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{TC(a)(,y), \beta(y,d)\}^{y}$
$\cup$
$\{\overline{\alpha}(c,d), \overline{\alpha}(c,e)\}^{\varnothing}, \{\overline{\alpha}(c,d), \overline{TC}(\overline{\alpha})(e,d)\}^{\varnothing}, \{TC(a)(c,y), \beta(y,d)\}^{y}$
$\overline{TC}$ •
$\{\overline{TC}(\overline{\alpha})(c,d)\}^{\varnothing}, \{TC(a)(c,y), \beta(y,d)\}^{y}$
$\vee,\exists,\wedge$
$\{\overline{\mathsf{ST}((aa \cup aba)^+)(c,d)} \vee \mathsf{ST}(a^+; ((ba^+)^+ \cup a))(c,d)\}^{\varnothing}$

Fig. 2. Cyclic proof for sequent not cyclically provable by TC_G.

5 Soundness of HTC

This section is devoted to the proof of the first of our main results:

Theorem 23 (Soundness). *If* $\mathsf{HTC} \vdash_{nwf} \mathbf{S}$ *then* $\models \mathbf{S}$.

The argument is quite technical due to the alternating nature of our progress condition. In particular the treatment of traces within hypertraces requires a more fine grained argument than usual, bespoke to our hypersequential structure.

Throughout this section, we shall fix a HTC preproof $\mathcal{D}$ of a hypersequent $\mathbf{S}$. For practical reasons we shall assume that $\mathcal{D}$ is substitution-free (at the cost of regularity) and that each quantifier in $\mathbf{S}$ binds a distinct variable.[4] We further assume some structure $\mathcal{M}^{\times}$ and an interpretation ρ_0 such that $\rho_0 \not\models \mathbf{S}$ (within $\mathcal{M}^{\times}$). Since each rule is locally sound, by contraposition we can continually choose 'false premisses' to construct an infinite 'false branch':

Lemma 24 (Countermodel branch). *There is a branch* $\mathcal{B}^{\times} = (\mathbf{S}_i)_{i<\omega}$ *of* $\mathcal{D}$ *and an interpretation* $\rho^{\times}$ *such that, with respect to* $\mathcal{M}^{\times}$*:*

[4] Note that this convention means we can simply take $y = x$ in the $\exists$ rule in Fig. 1.

1. *$\rho^\times \not\models \mathbf{S}_i$, for all $i < \omega$;*
2. *Suppose that $\mathbf{S}_i$ concludes a $\overline{TC}$ step, as typeset in Fig. 1, and $\rho^\times \models TC(\bar{A})(s,t)\,[\mathbf{d}/\mathbf{x}]$. If n is minimal such that $\rho^\times \models \bar{A}(d_i, d_{i+1})$ for all $i \leq n$, $\rho^\times(s) = d_0$ and $\rho^\times(t) = d_n$, and $n > 1$, then $\rho^\times(f)(\mathbf{d}) = d_1$[5] so that $\rho_{i+1} \models \bar{A}(s, f(\mathbf{x}))[\mathbf{d}/\mathbf{x}]$ and $\rho^\times \models TC(\bar{A})(f(\mathbf{x}), t)[\mathbf{d}/\mathbf{x}]$.*

Unpacking this a little, our interpretation $\rho^\times$ is actually defined as the limit of a chain of 'partial' interpretations $(\rho_i)_{i<\omega}$, with each $\rho_i \not\models \mathbf{S}_i$ (within $\mathcal{M}^\times$). Note in particular that, by 2, whenever some $\overline{TC}$-formula is principal, we choose ρ_{i+1} to always assign to it a falsifying path of minimal length (if one exists at all), with respect to the assignment to variables in its annotation. It is crucial at this point that our definition of $\rho^\times$ is parametrised by such assignments.

Let us now fix $\mathcal{B}^\times$ and $\rho^\times$ as provided by the Lemma above. Moreover, let us henceforth assume that $\mathcal{D}$ is a proof, i.e. it is progressing, and fix a progressing hypertrace $\mathcal{H} = (\{\Gamma_i\}^{\mathbf{x}_i})_{i<\omega}$ along $\mathcal{B}^\times$. In order to carry out an infinite descent argument, we will need to define a particular trace along this hypertrace that 'preserves' falsity, bottom-up. This is delicate since the truth values of formulas in a trace depend on the assignment of elements to variables in the annotations. A particular issue here is the instantiation rule inst, which requires us to 'revise' whatever assignment of y we may have defined until that point. Thankfully, our earlier convention on substitution-freeness and uniqueness of bound variables in $\mathcal{D}$ facilitates the convergence of this process to a canonical such assignment:

Definition 25 (Assignment). *We define $\delta_\mathcal{H} : \bigcup\limits_{i<\omega} \mathbf{x}_i \to |\mathcal{M}^\times|$ by $\delta_\mathcal{H}(x) := \rho(t)$ if x is instantiated by t in $\mathcal{H}$; otherwise $\delta_\mathcal{H}(x)$ is some arbitrary $d \in |\mathcal{M}^\times|$.*

Note that $\delta_\mathcal{H}$ is indeed well-defined, thanks to the convention that each quantifier in $\mathbf{S}$ binds a distinct variable. In particular we have that each variable x is instantiated at most once along a hypertrace. Henceforth we shall simply write $\rho, \delta_\mathcal{H} \models A(\mathbf{x})$ instead of $\rho \models A(\delta_\mathcal{H}(\mathbf{x}))$. Working with such an assignment ensures that false formulas along $\mathcal{H}$ always have a false immediate ancestor:

Lemma 26 (Falsity through $\mathcal{H}$). *If $\rho^\times, \delta_\mathcal{H} \not\models F$ for some $F \in \Gamma_i$, then F has an immediate ancestor $F' \in \Gamma_{i+1}$ with $\rho^\times, \delta_\mathcal{H} \not\models F'$.*

In particular, regarding the inst rule of Fig. 1, note that if $F \in \Gamma(y)$ then we can choose $F' = F[t/y]$ which, by definition of $\delta_\mathcal{H}$, has the same truth value. By repeatedly applying this Lemma we obtain:

Proposition 27 (False trace). *There exists an infinite trace $\tau^\times = (F_i)_{i<\omega}$ through $\mathcal{H}$ such that, for all i, it holds that $\mathcal{M}^\times, \rho^\times, \delta_\mathcal{H} \not\models F_i$.*

We are now ready to prove our main soundness result.

Proof (of Theorem 23, sketch). Fix the infinite trace $\tau^\times = (F_i)_{i<\omega}$ through $\mathcal{H}$ obtained by Proposition 27. Since $\tau^\times$ is infinite, by definition of HTC proofs, it

[5] To be clear, we here choose an arbitrary such minimal '$\bar{A}$-path'.

needs to be progressing, i.e., it is infinitely often $\overline{TC}$-principal and there is some $k \in \mathbb{N}$ s.t. for $i > k$ we have that $F_i = \overline{TC}(A)(s_i, t_i)$ for some terms s_i, t_i.

To each F_i, for $i > k$, we associate the natural number n_i measuring the '$\bar{A}$-distance between s_i and t_i'. Formally, $n_i \in \mathbb{N}$ is least such that there are $d_0, \dots, d_{n_i} \in |\mathcal{M}^\times|$ with $\rho^\times(s) = d_0, \rho^\times(t) = d_{n_i}$ and, for all $i < n_i$, $\rho^\times, \delta_{\mathcal{H}} \models \bar{A}(d_i, d_{i+1})$. Our aim is to show that $(n_i)_{i>k}$ has no minimal element, contradicting wellfoundness of $\mathbb{N}$. For this, we establish the following two local properties:

$$\mathsf{id}\,\frac{}{p,\bar{p}} \qquad \mathsf{wk}\,\frac{\Gamma}{\Gamma, A} \qquad \mathsf{k}_a\,\frac{\Gamma, A}{\langle a\rangle\Gamma, [a]A} \qquad \wedge\,\frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \wedge B} \qquad \vee_0\,\frac{\Gamma, A_0}{\Gamma, A_0 \vee A_1} \qquad \vee_1\,\frac{\Gamma, A_1}{\Gamma, A_0 \vee A_1}$$

$$\langle ; \rangle\,\frac{\Gamma, \langle\alpha\rangle\langle\beta\rangle A}{\Gamma, \langle\alpha;\beta\rangle A} \qquad \langle\cup\rangle_0\,\frac{\Gamma, \langle\alpha_0\rangle A}{\Gamma, \langle\alpha_0 \cup \alpha_1\rangle A} \qquad \langle\cup\rangle_1\,\frac{\Gamma, \langle\alpha_1\rangle A}{\Gamma, \langle\alpha_0 \cup \alpha_1\rangle A} \qquad [\cup]\,\frac{\Gamma, [\alpha]A \quad \Gamma, [\beta]A}{\Gamma, [\alpha \cup \beta]A}$$

$$[;]\,\frac{\Gamma, [\alpha][\beta]A}{\Gamma, [\alpha;\beta]A} \qquad \langle + \rangle_0\,\frac{\Gamma, \langle\alpha\rangle A}{\Gamma, \langle\alpha^+\rangle A} \qquad \langle + \rangle_1\,\frac{\Gamma, \langle\alpha\rangle\langle\alpha^+\rangle A}{\Gamma, \langle\alpha^+\rangle A} \qquad [+]\,\frac{\Gamma, [\alpha]A \quad \Gamma, [\alpha][\alpha^+]A}{\Gamma, [\alpha^+]A}$$

Fig. 3. Rules of LPD^+.

1. $(n_i)_{i>k}$ is *monotone decreasing*, i.e., for all $i > k$, we have $n_{i+1} \leq n_i$;
2. Whenever F_i is principal, we have $n_{i+1} < n_i$.

So $(n_i)_{i>k}$ is monotone decreasing, by 1, but cannot converge, by 2 and the definition of progressing trace. Thus $(n_i)_{k<i}$ has no minimal element, yielding the required contradiction.

6 HTC is Complete for PDL$^+$, Over Standard Translation

In this section we give our next main result:

Theorem 28 (Completeness for PDL$^+$**).** *For a* PDL$^+$ *formula A, if $\models A$ then* $\mathsf{HTC} \vdash_{cyc} \mathsf{ST}(A)(c)$.

The proof is by a direct simulation of a cut-free cyclic system for PDL$^+$ that is complete. We shall briefly sketch this system below.

6.1 Circular System for PDL$^+$

The system LPD^+, given in Fig. 3, is the natural extension of the usual sequent calculus for basic multimodal logic K by rules for programs. In Fig. 3, $\langle a\rangle\Gamma$ is shorthand for $\{\langle a\rangle B : B \in \Gamma\}$. (Regular) preproofs for this system are defined just like for HTC or TC_G. The notion of 'immediate ancestor' is induced by the indicated colouring: a formula C in a premiss is an immediate ancestor of a formula C' in the conclusion if they have the same colour; if $C, C' \in \Gamma$ then we furthermore require $C = C'$.

Definition 29 (Non-wellfounded proofs). *Fix a preproof $\mathcal{D}$ of a sequent Γ. A* thread *is a maximal path in its graph of immediate ancestry. We say a thread is* progressing *if it has a smallest infinitely often principal formula of the form $[\alpha^+]A$. $\mathcal{D}$ is a* proof *if every infinite branch has a progressing thread. If $\mathcal{D}$ is regular, we call it a* cyclic proof *and we may write $\mathsf{LPD}^+ \vdash_{cyc} \Gamma$.*

Soundness of cyclic-LPD^+ is established by a standard infinite descent argument, but is also implied by the soundness of cyclic-HTC (Theorem 23) and the simulation we are about to give (Theorem 28), though this is somewhat overkill. Completeness may be established by the game theoretic approach of Niwinskí and Walukiewicz [23], as done by Lange [20] for PDL (with identity), or by purely proof theoretic techniques of Studer [25]. Either way, both results follow from a standard embedding of PDL$^+$ into the μ-calculus and its known completeness results [23,25], by way of a standard 'proof reflection' argument: μ-calculus proofs of the embedding are 'just' step-wise embeddings of LPD^+ proofs:

Theorem 30 (Soundness and completeness, [20]). *Let A be a* PDL$^+$ *formula. $\models A$ iff $\mathsf{LPD}^+ \vdash_{cyc} A$.*

6.2 A 'Local' Simulation of LPD$^+$ by HTC

In this subsection we show that LPD^+-preproofs can be stepwise transformed into HTC-proofs, with respect to the standard translation. In order to produce this local simulation, we need a more refined version of the standard translation that incorporates the structural elements of hypersequents.

Fix a PDL$^+$ formula $A = [\alpha_1]\dots[\alpha_n]\langle\beta_1\rangle\dots\langle\beta_m\rangle B$, for $n, m \geq 0$. The *hypersequent translation* of A, written $\mathsf{HT}(A)(c)$, is defined as:

$$
\begin{array}{l}
\{\overline{\mathsf{ST}(\alpha_1)(c,d_1)}\}^{\varnothing}, \{\overline{\mathsf{ST}(\alpha_2)(d_1,d_2)}\}^{\varnothing}, \dots, \{\overline{\mathsf{ST}(\alpha_n)(d_{n-1},d_n)}\}^{\varnothing},\\
\{\mathsf{ST}(\beta_1)(d_n,y_1), \mathsf{ST}(\beta_2)(y_2,y_3), \dots, \mathsf{ST}(\beta_m)(y_{m-1},y_m), \mathsf{ST}(B)(y_m)\}^{y_1,\dots,y_m}
\end{array}
$$

For $\Gamma = A_1, \dots, A_k$, we write $\mathsf{HT}(\Gamma)(c) := \mathsf{HT}(A_1)(c), \dots, \mathsf{HT}(A_k)(c)$.

Definition 31 (HT-translation). *Let $\mathcal{D}$ be a* PDL$^+$ *preproof. We shall define a HTC preproof $\mathsf{HT}(\mathcal{D})(c)$ of the hypersequent $\mathsf{HT}(A)(c)$ by a local translation of inference steps. We give only a few of the important cases here, but a full definition can be found in [13].*

- *A step* $\mathsf{k}_a \, \dfrac{B_1, \dots, B_k, A}{\langle a\rangle B_1, \dots, \langle a\rangle B_k, [a]A}$ *is translated to:*

$$
\begin{array}{rc}
 & \mathsf{HT}(B_1)(c), \dots, \mathsf{HT}(B_k)(c), \mathsf{HT}(A)(c) \\ \hline
{[d/c]} & \mathsf{HT}(B_1)(d), \dots, \mathsf{HT}(B_k)(d), \mathsf{HT}(A)(d) \\ \hline\hline
\vee,\forall & \{\mathsf{CT}(B_1)(d)\}^{\mathbf{x}_{B_1}}, \dots, \{\mathsf{CT}(B_k)(d)\}^{\mathbf{x}_{B_k}}, \mathsf{HT}(A)(d) \\ \hline
\mathsf{wk} & \{\mathsf{CT}(B_1)(d)\}^{\mathbf{x}_{B_1}}, \dots, \{\mathsf{CT}(B_k)(d)\}^{\mathbf{x}_{B_k}}, \{\overline{\mathsf{ST}(a)(c,d)}\}^{\varnothing}, \mathsf{HT}(A)(d) \\ \hline\hline
\cup & \{\mathsf{CT}(B_1)(d)\}^{\mathbf{x}_{B_1}}, \dots, \{\mathsf{ST}(a)(c,d), \mathsf{CT}(B_k)(d)\}^{\mathbf{x}_{B_k}}, \{\overline{\mathsf{ST}(a)(c,d)}\}^{\varnothing}, \mathsf{HT}(A)(d) \\ \hline\hline
\mathsf{inst} & \{\mathsf{ST}(a)(c,y), \mathsf{CT}(B_1)(y)\}^{\mathbf{x}_{B_1},y}, \dots, \{\mathsf{ST}(a)(c,y), \mathsf{CT}(B_k)(y)\}^{\mathbf{x}_{B_k},y}, \{\overline{\mathsf{ST}(a)(c,d)}\}^{\varnothing}, \mathsf{HT}(A)(d) \\ \hdashline
= & \mathsf{HT}(\langle a\rangle B_1)(c), \dots, \mathsf{HT}(\langle a\rangle B_k)(c), \mathsf{HT}([a]A)(c)
\end{array}
$$

where (omitted) left-premisses of $\cup$ *steps are simply proved by* $\mathsf{wk}, \mathsf{id}, \mathsf{init}$*. In this and the following cases, we use the notation* $\mathsf{CT}(A)(c)$ *and* $\mathbf{x}_A$ *for the appropriate sets of formulas and variables forced by the definition of* HT *(again, see [13] for further details).*

- *A* $\langle\cup\rangle_i$ *step (for* $i = 0, 1$*), as typeset in Fig. 3, is translated to:*

$$= \dfrac{\vee\,\dfrac{=\,\dfrac{\mathsf{HT}(\Gamma)(c), \mathsf{HT}(\langle\alpha_i\rangle A)(c)}{\mathsf{HT}(\Gamma)(c), \{\mathsf{ST}(\alpha_i)(c, y), \mathsf{CT}(A)(y)\}^{\mathbf{x}_B, y}}}{\mathsf{HT}(\Gamma)(c), \{\mathsf{ST}(\alpha_0)(c, y) \vee \mathsf{ST}(\alpha_1)(c, y), \mathsf{CT}(A)(y)\}^{\mathbf{x}_A, y}}}{\mathsf{HT}(\Gamma)(c), \mathsf{HT}(\langle\alpha_0 \cup \alpha_1\rangle A)(c)}$$

- *A* $\langle ; \rangle$ *step, as typeset in Fig. 3, is translated to:*

$$= \dfrac{\exists\,\dfrac{\wedge\,\dfrac{=\,\dfrac{\mathsf{HT}(\Gamma)(c), \mathsf{HT}(\langle\alpha\rangle\langle\beta\rangle A)(c)}{\mathsf{HT}(\Gamma)(c), \{\mathsf{ST}(\alpha)(c, z), \mathsf{ST}(\alpha)(z, y), \mathsf{CT}(A)(y)\}^{\mathbf{x}_A, y, z}}}{\mathsf{HT}(\Gamma)(c), \{\mathsf{ST}(\alpha)(c, z) \wedge \mathsf{ST}(\alpha)(z, y), \mathsf{CT}(A)(y)\}^{\mathbf{x}_A, y, z}}}{\mathsf{HT}(\Gamma)(c), \{\exists z(\mathsf{ST}(\alpha)(c, z) \wedge \mathsf{ST}(\alpha)(z, y)), \mathsf{CT}(A)(y)\}^{\mathbf{x}_A, y}}}{\mathsf{HT}(\Gamma)(c), \mathsf{HT}(\langle\alpha; \beta\rangle A)(c)}$$

- *A* $[+]$ *step, as typeset in Fig. 3, is translated to:*

$$= \dfrac{\overline{TC}\,\dfrac{\cup\,\dfrac{\mathcal{E} \qquad \cup\,\dfrac{\mathcal{E}' \qquad =\,\dfrac{\mathsf{HT}(\Gamma)(c), \mathsf{HT}([\alpha][\alpha^+]A)(c)}{\mathsf{HT}(\Gamma)(c), \{\overline{\mathsf{ST}(\alpha)(c, f)}\}^{\varnothing}, \{\overline{TC}(\overline{\mathsf{ST}(\alpha)})(f, d)\}^{\varnothing}, \mathsf{HT}(A)(d)}}{\mathsf{HT}(\Gamma)(c), \{\overline{\mathsf{ST}(\alpha)(c, f)}\}^{\varnothing}, \{\overline{\mathsf{ST}(\alpha)(c, d)}, \overline{TC}(\overline{\mathsf{ST}(\alpha)})(f, d)\}^{\varnothing}, \mathsf{HT}(A)(d)}}{\mathsf{HT}(\Gamma)(c), \{\overline{\mathsf{ST}(\alpha)(c, d)}, \overline{\mathsf{ST}(\alpha)(c, f)}\}^{\varnothing}, \{\overline{\mathsf{ST}(\alpha)(c, d)}, \overline{TC}(\overline{\mathsf{ST}(\alpha)})(f, d)\}^{\varnothing}, \mathsf{HT}(A)(d)}}{\mathsf{HT}(\Gamma)(c), \{\overline{TC}(\overline{\mathsf{ST}(\alpha)})(c, d)\}^{\varnothing}, \mathsf{HT}(A)(d)}}{\mathsf{HT}(\Gamma)(c), \mathsf{HT}([\alpha^+]A)(c)}$$

where $\mathcal{E}$ *and* $\mathcal{E}'$ *derive* $\mathsf{HT}(\Gamma)(c)$ *and* $\mathsf{HT}([\alpha]A)(c)$*, resp., using* wk*-steps.*

Note that, formally speaking, the well-definedness of $\mathsf{HT}(\mathcal{D})(c)$ in the definition above is guaranteed by coinduction: each rule of $\mathcal{D}$ is translated into a (nonempty) derivation.

Remark 32 (Deeper inference). Observe that HTC can also simulate 'deeper' program rules than are available in LPD^+. E.g. a rule $\frac{\Gamma, \langle\alpha\rangle\langle\beta_i\rangle A}{\Gamma, \langle\alpha\rangle\langle\beta_0 \cup \beta_1\rangle A}$ may be simulated too (similarly for $[\,]$). E.g. $\langle a^+\rangle\langle b\rangle p \supset \langle a^+\rangle\langle b \cup c\rangle p$ admits a *finite* proof in HTC (under ST), rather than a necessarily infinite (but cyclic) one in LPD^+.

6.3 Justifying Regularity and Progress

Proposition 33. *If* $\mathcal{D}$ *is regular, then so is* $\mathsf{HT}(\mathcal{D})(c)$*.*

Proof. Notice that each rule in $\mathcal{D}$ is translated to a finite derivation in $\mathsf{HT}(\mathcal{D})(c)$. Thus, if $\mathcal{D}$ has only finitely many distinct subproofs, then also $\mathsf{HT}(\mathcal{D})(c)$ has only finitely many distinct subproofs.

Proposition 34. *If $\mathcal{D}$ is progressing, then so is $\mathsf{HT}(\mathcal{D})(c)$.*

Proof (sketch). We need to show that every infinite branch of $\mathsf{HT}(\mathcal{D})(c)$ has a progressing hypertrace. Since the HT translation is defined stepwise on the individual steps of $\mathcal{D}$, we can associate to each infinite branch $\mathcal{B}$ of $\mathsf{HT}(\mathcal{D})(c)$ a unique infinite branch $\mathcal{B}'$ of $\mathcal{D}$. Since $\mathcal{D}$ is progressing, let $\tau = (F_i)_{i<\omega}$ be a progressing thread along $\mathcal{B}'$. By inspecting the rules of LPD^+ (and by definition of progressing thread), for some $k \in \mathbb{N}$, each F_i for $i > k$ has the form: $[\alpha_{i,1}] \cdots [\alpha_{i,n_i}][\alpha^+]A$, for some $n_i \geq 0$. So, for $i > k$, $\mathsf{HT}(F_i)(d_i)$ has the form:

$$\{\overline{\mathsf{ST}(\alpha_{i,1})(c, d_{i,1})}\}^{\varnothing}, \dots, \{\overline{\mathsf{ST}(\alpha_{i,n_i})(d_{i,n_i-1}, d_{i,n_i})}\}^{\varnothing}, \{\overline{TC}(\overline{\mathsf{ST}(\alpha)})(d_{i,n_i}, d_i)\}^{\varnothing}, \mathsf{HT}(A)(d_i)$$

By inspection of the HT-translation (Definition 31) whenever F_{i+1} is an immediate ancestor of F_i in $\mathcal{B}'$, there is a path from the cedent $\{\overline{TC}(\overline{\mathsf{ST}(\alpha)})(d_{i+1,n_{i+1}}, d_{i+1})\}^{\varnothing}$ to the cedent $\{\overline{TC}(\overline{\mathsf{ST}(\alpha)})(d_{i,n_i}, d_i)\}^{\varnothing}$ in the graph of immediate ancestry along $\mathcal{B}$. Thus, since $\tau = (F_i)_{i<\omega}$ is a trace along $\mathcal{B}'$, we have a (infinite) hypertrace of the form $\mathcal{H}_\tau := (\{\Delta_i, \overline{TC}(\overline{\mathsf{ST}(\alpha)})(d_{i,n_i}, d_i)\}^{\varnothing})_{i>k'}$ along $\mathcal{B}$. By construction $\Delta_i = \varnothing$ for infinitely many $i > k'$, and so $\mathcal{H}_\tau$ has just one infinite trace. Moreover, by inspection of the $[+]$ step in Definition 31, this trace progresses in $\mathcal{B}$ every time τ does in $\mathcal{B}'$, and so progresses infinitely often. Thus, $\mathcal{H}$ is a progressing hypertrace. Since the choice of the branch $\mathcal{B}$ of $\mathcal{D}$ was arbitrary, we are done.

6.4 Putting it all Together

We can now finally conclude our main simulation theorem:

Proof (of Theorem 28, sketch). Let A be a PDL^+ formula s.t. $\models A$. By the completeness result for LPD^+, Theorem 30, we have that $\mathsf{LPD}^+ \vdash_{cyc} A$, say by a cyclic proof $\mathcal{D}$. From here we construct the HTC preproof $\mathsf{HT}(\mathcal{D})(c)$ which, by Propositions 33 and 34, is in fact a cyclic proof of $\mathsf{HT}(A)(c)$. Finally, we apply some basic $\vee, \wedge, \exists, \forall$ steps to obtain a cyclic HTC proof of $\mathsf{ST}(A)(c)$.

7 Extension by Equality and Simulating Full PDL

We now briefly explain how our main results are extended to the 'reflexive' version of TCL. The language of $\mathsf{HTC}_=$ allows further atomic formulas of the form $s = t$ and $s \neq t$. The calculus $\mathsf{HTC}_=$ extends HTC by the rules:

$$= \frac{\mathbf{S}, \{\Gamma\}^{\mathbf{x}}}{\mathbf{S}, \{t = t, \Gamma\}^{\mathbf{x}}} \qquad \neq \frac{\mathbf{S}, \{\Gamma(s), \Delta(s)\}^{\mathbf{x}}}{\mathbf{S}, \{\Gamma(s), s \neq t\}^{\mathbf{x}}, \{\Delta(t)\}^{\mathbf{x}}}$$

The notion of immediate ancestry is colour-coded as in Definition 15, and the resulting notions of (pre)proof, (hyper)trace and progress are as in Definition 17. The simulation of Cohen and Rowe's system TC_G extends to their reflexive system, RTC_G, by defining their operator $RTC(\lambda x, y.A)(s,t) := TC(\lambda x, y.(x = y \vee A))(s,t)$. Note that, while it is semantically correct to set $RTC(A)(s,t)$ to be $s = t \vee TC(A)(s,t)$, this encoding does not lift to the Cohen-Rowe rules for RTC. Understanding that structures interpret $=$ as true equality, a modular adaptation of the soundness argument for HTC, cf. Sect. 5, yields:

Theorem 35 (Soundness of $\mathsf{HTC}_=$). *If* $\mathsf{HTC}_= \vdash_{nwf} \mathbf{S}$ *then* $\models \mathbf{S}$*.*

Turning to the modal setting, PDL may be defined as the extension of PDL^+ by including a program $A?$ for each formula A. Semantically, we have $(A?)^{\mathcal{M}} = \{(v,v) : \mathcal{M}, v \models A\}$. From here we may define $\varepsilon := \top?$ and $\alpha^* := (\varepsilon \cup \alpha)^+$; again, while it is semantically correct to set $\alpha^* = \varepsilon \cup \alpha^+$, this encoding does not lift to the standard sequent rules for $*$. The system LPD is obtained from LPD^+ by including the rules:

$$\langle ? \rangle \frac{\Gamma, A \quad \Gamma, B}{\Gamma, \langle A? \rangle B} \qquad [?] \frac{\Gamma, \bar{A}, B}{\Gamma, [A?]B}$$

Again, the notion of immediate ancestry is colour-coded as for LPD^+; the resulting notions of (pre)proof, thread and progress are as in Definition 29. Just like for LPD^+, a standard encoding of LPD into the μ-calculus yields its soundness and completeness, thanks to known sequent systems for the latter, cf. [23,25], but has also been established independently [20]. Again, a modular adaptation of the simulation of LPD^+ by HTC, cf. Sect. 6, yields:

Theorem 36 (Completeness for PDL). *Let A be a* PDL *formula. If* $\models A$ *then* $\mathsf{HTC}_= \vdash_{cyc} \mathsf{ST}(A)(c)$*.*

8 Conclusions

In this work we proposed a novel cyclic system HTC for Transitive Closure Logic (TCL) based on a form of hypersequents. We showed a soundness theorem for standard semantics, requiring an argument bespoke to our hypersequents. Our system is cut-free, rendering it suitable for automated reasoning via proof search. We showcased its expressivity by demonstrating completeness for PDL, over the standard translation. In particular, we demonstrated formally that such expressivity is not available in the previously proposed system TC_G of Cohen and Rowe (Theorem 12). Our system HTC locally simulates TC_G too (Theorem 19).

As far as we know, HTC is the first cyclic system employing a form of *deep inference* resembling *alternation* in automata theory, e.g. wrt. proof checking, cf. Proposition 18. It would be interesting to investigate the structural proof theory that emerges from our notion of hypersequent. As hinted at in Examples 20 and 21, our hypersequential system exhibits more liberal rule permutations than usual sequents, so we expect their *focussing* and *cut-elimination* behaviours to similarly be richer, cf. [21,22]. Note however that such investigations are rather pertinent for pure predicate logic (without TC): focussing and cut-elimination arguments do not typically preserve regularity of non-wellfounded proofs, cf. [2].

Finally, our work bridges the cyclic proof theories of (identity-free) PDL and (reflexive) TCL. With increasing interest in both modal and predicate cyclic proof theory, it would be interesting to further develop such correspondences.

Acknowledgements. The authors would like to thank Sonia Marin, Jan Rooduijn and Reuben Rowe for helpful discussions on matters surrounding this work.

References

1. Avron, A.: Transitive closure and the mechanization of mathematics. In: Kamareddine, F.D. (eds) Thirty Five Years of Automating Mathematics. Applied Logic Series, vol. 28, pp. 149–171. Springer, Dordrecht (2003). https://doi.org/10.1007/978-94-017-0253-9_7
2. Baelde, D., Doumane, A., Saurin, A.: Infinitary proof theory: the multiplicative additive case. In: Talbot, J., Regnier, L. (eds.) 25th EACSL Annual Conference on Computer Science Logic, CSL 2016, 29 August–1 September 2016, Marseille, France. LIPIcs, vol. 62, pp. 42:1–42:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.CSL.2016.42
3. Berardi, S., Tatsuta, M.: Classical system of martin-lof's inductive definitions is not equivalent to cyclic proofs. CoRR abs/1712.09603 (2017). http://arxiv.org/abs/1712.09603
4. Blackburn, P., van Benthem, J.: Modal logic: a semantic perspective. In: Blackburn, P., van Benthem, J.F.A.K., Wolter, F. (eds.) Handbook of Modal Logic, Studies in Logic and Practical Reasoning, vol. 3, pp. 1–84. North-Holland (2007). https://doi.org/10.1016/s1570-2464(07)80004-8
5. Blackburn, P., De Rijke, M., Venema, Y.: Modal Logic, vol. 53. Cambridge University Press (2002)
6. Brotherston, J., Distefano, D., Petersen, R.L.: Automated cyclic entailment proofs in separation logic. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS (LNAI), vol. 6803, pp. 131–146. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22438-6_12
7. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: Jhala, R., Igarashi, A. (eds.) APLAS 2012. LNCS, vol. 7705, pp. 350–367. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35182-2_25
8. Brotherston, J., Simpson, A.: Sequent calculi for induction and infinite descent. J. Log. Comput. **21**(6), 1177–1216 (2011)
9. Cohen, L., Rowe, R.N.S.: Uniform inductive reasoning in transitive closure logic via infinite descent. In: Ghica, D.R., Jung, A. (eds.) 27th EACSL Annual Conference on Computer Science Logic, CSL 2018, 4–7 September 2018, Birmingham, UK. LIPIcs, vol. 119, pp. 17:1–17:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). https://doi.org/10.4230/LIPIcs.CSL.2018.17
10. Cohen, L., Rowe, R.N.S.: Integrating induction and coinduction via closure operators and proof cycles. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12166, pp. 375–394. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51074-9_21
11. Cohen, L., Rowe, R.N.: Non-well-founded proof theory of transitive closure logic. ACM Trans. Comput. Log. **21**(4), 1–31 (2020)
12. Das, A.: On the logical complexity of cyclic arithmetic. Log. Methods Comput. Sci. **16**(1) (2020). https://doi.org/10.23638/LMCS-16(1:1)2020

13. Das, A., Girlando, M.: Cyclic proofs, hypersequents, and transitive closure logic (2022). https://doi.org/10.48550/ARXIV.2205.08616
14. Das, A., Pous, D.: A cut-free cyclic proof system for Kleene algebra. In: Schmidt, R.A., Nalon, C. (eds.) TABLEAUX 2017. LNCS (LNAI), vol. 10501, pp. 261–277. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66902-1_16
15. Grädel, E.: On transitive closure logic. In: Börger, E., Jäger, G., Kleine Büning, H., Richter, M.M. (eds.) CSL 1991. LNCS, vol. 626, pp. 149–163. Springer, Heidelberg (1992). https://doi.org/10.1007/BFb0023764
16. Gurevich, Y.: Logic and the Challenge of Computer Science, pp. 1–57. Computer Science Press (1988). https://www.microsoft.com/en-us/research/publication/logic-challenge-computer-science/
17. Immerman, N.: Languages that capture complexity classes. SIAM J. Comput. **16**(4), 760–778 (1987). https://doi.org/10.1137/0216051
18. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. In: Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS 1991), Amsterdam, The Netherlands, 15–18 July 1991, pp. 214–225. IEEE Computer Society (1991). https://doi.org/10.1109/LICS.1991.151646
19. Krob, D.: Complete systems of b-rational identities. Theor. Comput. Sci. **89**(2), 207–343 (1991). https://doi.org/10.1016/0304-3975(91)90395-I
20. Lange, M.: Games for modal and temporal logics. Ph.D. thesis (2003)
21. Marin, S., Miller, D., Volpe, M.: A focused framework for emulating modal proof systems. In: Beklemishev, L.D., Demri, S., Maté, A. (eds.) Advances in Modal Logic 11, Proceedings of the 11th Conference on "Advances in Modal Logic," held in Budapest, Hungary, 30 August–2 September 2016, pp. 469–488. College Publications (2016). http://www.aiml.net/volumes/volume11/Marin-Miller-Volpe.pdf
22. Miller, D., Volpe, M.: Focused labeled proof systems for modal logic. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) LPAR 2015. LNCS, vol. 9450, pp. 266–280. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48899-7_19
23. Niwiński, D., Walukiewicz, I.: Games for the mu-calculus. Theor. Comput. Sci. **163**(1), 99–116 (1996). https://doi.org/10.1016/0304-3975(95)00136-0
24. Rowe, R.N.S., Brotherston, J.: Automatic cyclic termination proofs for recursive procedures in separation logic. In: Bertot, Y., Vafeiadis, V. (eds.) Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, 16–17 January 2017, pp. 53–65. ACM (2017). https://doi.org/10.1145/3018610.3018623
25. Studer, T.: On the proof theory of the modal mu-calculus. Stud. Logica. **89**(3), 343–363 (2008)
26. Tellez, G., Brotherston, J.: Automatically verifying temporal properties of pointer programs with cyclic proof. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 491–508. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_30
27. Tellez, G., Brotherston, J.: Automatically verifying temporal properties of pointer programs with cyclic proof. J. Autom. Reason. **64**(3), 555–578 (2020). https://doi.org/10.1007/s10817-019-09532-0

Equational Unification and Matching, and Symbolic Reachability Analysis in Maude 3.2 (System Description)

Francisco Durán[1], Steven Eker[2], Santiago Escobar[3](✉), Narciso Martí-Oliet[4], José Meseguer[5], Rubén Rubio[4], and Carolyn Talcott[2]

[1] Universidad de Málaga, Málaga, Spain
duran@lcc.uma.es
[2] SRI International, Menlo Park, CA, USA
eker@csl.sri.com, clt@cs.stanford.edu
[3] VRAIN, Universitat Politècnica de València, Valencia, Spain
sescobar@upv.es
[4] Universidad Complutense de Madrid, Madrid, Spain
{narciso,rubenrub}@ucm.es
[5] University of Illinois at Urbana-Champaign, Urbana, IL, USA
meseguer@illinois.edu

Abstract. Equational unification and matching are fundamental mechanisms in many automated deduction applications. Supporting them efficiently for as wide as possible a class of equational theories, and in a typed manner supporting type hierarchies, benefits many applications; but this is both challenging and nontrivial. We present Maude 3.2's efficient support of these features as well as of symbolic reachability analysis of infinite-state concurrent systems based on them.

1 Introduction

Unification is a key mechanism in resolution [41] and paramodulation-based [36] theorem proving. Since Plotkin's work [40] on *equational unification*, i.e.,

Durán was supported by the grant UMA18-FEDERJA-180 funded by J. Andalucía/FEDER and the grant PGC2018-094905-B-I00 funded by MCIN/AEI/10.13039/501100011033 and ERDF A way of making Europe. Escobar was supported by the EC H2020-EU grant 952215, by the grant RTI2018-094403-B-C32 funded by MCIN/AEI/10.13039/501100011033 and ERDF A way of making Europe, by the grant PROMETEO/2019/098 funded by Generalitat Valenciana, and by the grant PCI2020-120708-2 funded by MICIN/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR. Martí-Oliet and Rubio were supported by the grant PID2019-108528RB-C22 funded by MCIN/AEI/10.13039/501100011033 and ERDF A way of making Europe. Talcott was partially supported by the U. S. Office of Naval Research under award numbers N00014-15-1-2202 and N00014-20-1-2644, and NRL grant N0017317-1-G002.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 529–540, 2022.
https://doi.org/10.1007/978-3-031-10769-6_31

E-unification modulo an equational theory E, it is widely used for increased effectiveness. Since Walther's work [47] it has been well understood that *typed* E-unification, exploiting types and subtype hierarchies, can drastically reduce a prover's search space. Many other automated deduction applications use typed E-unification as a key mechanism, including, inter alia: (i) constraint logic programming, e.g., [12,23]; (ii) narrowing-based infinite-state reachability analysis and model checking, e.g., [6,35]; (iii) cryptographic protocol analysis modulo algebraic properties, e.g., [8,19,28]; (iv) partial evaluation, e.g., [4,5]; and (v) SMT solving, e.g., [32,48]. The special case of typed *E-matching* is also a key component in all the above areas as well as in: (vi) E-generalization (also called anti-unification), e.g., [1,2]; and (vii) E-homeomorphic embedding, e.g., [3].

Maximizing the scope and effectiveness of typed E-unification and E-matching means efficiently supporting as wide a class of theories E as possible. Such efficiency crucially depends on both efficient algorithms (and their combinations) and —since the number of E-unifiers may be large— on computing complete *minimal* sets of solutions to reduce the search space. The recent Maude 3.2 release[1] provides this kind of efficient support for typed E-unification and E-matching in three, increasingly more general classes of theories E:

1. Typed B-unification and B-matching, where B is any combination of associativity (A) and/or commutativity (C) and/or unit element (U) axioms.
2. Typed $E \cup B$-unification and matching in the *user-definable* infinite class of theories $E \cup B$ with B as in (1), and $E \cup B$ having the *finite variant property* (FVP) [13,21].
3. Typed $E \cup B$-unification for the infinite class of *user-definable* theories $E \cup B$ with B as in (1), and E confluent, terminating, and coherent modulo B.

For classes (1) and (2) the set of B- (resp. $E \cup B$-) unifiers is always *complete, minimal and finite*, except for the *AwoC* case when B contains an A but not C axiom for some binary symbol f.[2] The typing is order-sorted [22,29] and thus contains many-sorted and unsorted B- (resp. $E \cup B$-) unification as special cases. For class (3), Maude enumerates a possibly infinite complete set of $E \cup B$-unifiers, with the same AwoC exception on B. We discuss new features for classes (1)–(2), and a new narrowing modulo $E \cup B$-based *symbolic reachability analysis* feature for infinite-state systems specified in Maude as rewrite theories $(\Sigma, E \cup B, R)$ with equations $E \cup B$ in class (2) and concurrent transition rules R. In Sect. 5 we discuss various applications that can benefit from these new features.

In comparison with previous Maude tool papers reporting on new features —the last one was [16]— the new features reported here include: (i) computing minimal complete sets of most general B- (resp. $E \cup B$-) unifiers for classes (1) and (2) except for the AwoC case; (ii) a new $E \cup B$-matching algorithm for class (2); and (iii) a new symbolic reachability analysis for concurrent systems

[1] Publicly available at http://maude.cs.illinois.edu.

[2] In the AwoC case, Maude's algorithms are optimized to favor many commonly occurring cases where typed A-unification is finitary, and provides a finite set of solutions and an incompleteness warning outside such cases (see [18]).

based on narrowing with transition rules modulo equations $E \cup B$ in class (2) enjoying powerful state-space reduction capabilities based on the minimality and completeness feature (i) and on "folding" less general symbolic states into more general ones through subsumption. Section 3.1 shows the importance of the new $E \cup B$-matching algorithm for efficient computation of minimal $E \cup B$-unifiers.

Notation, Strict-B-Coherence, and FVP. For notation involving either term positions, $p \in pos(t)$, $t|_p$, $t[t']_p$, or substitutions, $t\theta$, $\theta\mu$, see [14]. Equations $(u = v) \in E$ oriented as rules $(u \to v) \in \overrightarrow{E}$ are *strictly coherent* modulo axioms B iff $(t =_B t' \wedge t \to_{\overrightarrow{E},B} w) \Rightarrow \exists w'(t \to_{\overrightarrow{E},B} w' \wedge w =_B w')$, where $t \to_{\overrightarrow{E},B} w$ iff $\exists (u \to v) \in \overrightarrow{E}$, $\exists \theta$, $\exists p \in pos(t)(u\theta =_B t|_p \wedge w = t[v\theta]_p)$. For $(\Sigma, E \cup B)$ an equational theory with $\overrightarrow{E}$ confluent, terminating and strictly coherent modulo B, (1) an $\overrightarrow{E}, B$-*t-variant* is a pair (v, θ) s.t. $v = (t\theta)!_{\overrightarrow{E},B} \wedge \theta = \theta!_{\overrightarrow{E},B}$, where $u!_{\overrightarrow{E},B}$ (resp. $\theta!_{\overrightarrow{E},B}$) denotes the $\overrightarrow{E}, B$-normal form of u, resp. θ; (2) for $\overrightarrow{E}, B$-t-variants (v, θ), (u, μ), the *more general relation* $(v, \theta) \sqsupseteq_B (u, \mu)$ holds iff $\exists \gamma (u =_B v\gamma \wedge \theta\gamma =_B \mu)$; (3) $(\Sigma, E \cup B)$ is *FVP* [13,21] iff any Σ-term t has a *finite* set of most general $\overrightarrow{E}, B$-t-variants. Footnote 5 explains how FVP can be checked.

2 Complete and Minimal Order-Sorted B-Unifiers

Throughout the paper we use the following equational theory $E \cup B$ of the Booleans as a running example (with self-explanatory, user-definable syntax[3]):

```
fmod BOOL-FVP is protecting TRUTH-VALUE .
   op _and_ : Bool Bool -> Bool [assoc comm] .
   op _xor_ : Bool Bool -> Bool [assoc comm] .
   op not_ : Bool -> Bool .
   op _or_ : Bool Bool -> Bool .
   op _<=>_ : Bool Bool -> Bool .
   vars X Y Z W : Bool .

   eq X and true = X [variant] .
   eq X and false = false [variant] .
   eq X and X = X  [variant] .
   eq X and X and Y = X and Y [variant] .   *** AC extension
   eq X xor false = X [variant] .
   eq X xor X = false [variant] .
   eq X xor X xor Y = Y [variant] .         *** AC extension
   eq not X = X xor true [variant] .
   eq X or Y = (X and Y) xor X xor Y [variant] .
   eq X <=> Y = true xor X xor Y [variant] .
endfm
```

[3] This module imports Maude's `TRUTH-VALUE` module and the command "`set include BOOL off .`" must be typed before the module to avoid default importation of `BOOL`.

The axioms B are the associativity-commutativity (AC) axioms for `xor` and `and` (specified with the `assoc comm` attributes). The equations E are terminating and confluent modulo B [42]. To achieve strict B-*coherence* [30], the needed AC-extensions [39] are added —for example, the AC-extension of `X xor X = false` is `X xor X xor Y = Y`. The equations E for `xor` and `and` define the theory of *Boolean rings*, *except for the missing*[4] *distributivity equation* `X and (Y xor Z) = (X and Y) xor (X and Z)`. The remaining equations in E define `or`, `not` and `<=>` as definitional extensions. The `variant` attribute declares that the equation will be used for folding variant narrowing [21]. The theory is FVP,[5] in class (2). In this section we will consider B-unification (for $B = AC$) using this example. $E \cup B$-unification for the same example will be discussed in Sect. 3.

For B any combination of associativity and/or commutativity and/or identity axioms, Maude's `unify` command computes a complete finite set of most general B-unifiers, except for the AwoC case. The new `irredundant unify` command always returns[6] a *finite, complete and minimal* set of B-unifiers, except for the AwoC case. The output of `unify` for the equation below can be found in [10, §13].

```
Maude> irredundant unify X and not Y and not Z =? W and Y and not X .
Decision time: 0ms cpu (0ms real)

Unifier 1                                Unifier 2
X --> #1:Bool and #2:Bool                X --> #2:Bool
Z --> #1:Bool and #2:Bool                Z --> #1:Bool
Y --> #1:Bool                            Y --> #2:Bool
W --> #2:Bool and not #1:Bool            W --> not #1:Bool
```

3 $E \cup B$-Unification and Matching for FVP Theories

It is a general result from [21] that if $E \cup B$ is FVP and B-unification is finitary, then $E \cup B$-unification is *finitary* and a complete finite set of $E \cup B$-unifiers can be computed by *folding variant narrowing* [21]. Furthermore, assuming that $T_{\Sigma/E,s}$ is non-empty for each sort s, a finitary $E \cup B$-unification algorithm automatically provides a decision procedure for *satisfiability* of any *positive* (the $\wedge, \vee$-fragment) quantifier-free formula φ in the initial algebra $T_{\Sigma/E}$, since φ can be put in DNF, and a conjunction of equalities Γ is satisfiable in $T_{\Sigma/E}$ iff Γ is $E \cup B$-unifiable.

Since for our running example `BOOL-FVP` the equations $E \cup B$ are FVP and B-unification (in this case $B = AC$) is finitary, all this has useful consequences for

[4] By missing distributivity, this theory is *weaker* than the theory of Boolean rings. Nevertheless, its *initial algebra* $T_{\Sigma/E \cup B}$ is exactly the Booleans on {`true,false`} with the standard truth tables for all connectives. Thus, all equations provable in Boolean algebra hold in $T_{\Sigma/E \cup B}$, including the missing distributivity equation.

[5] This can be easily checked in Maude by checking the finiteness of the variants for each $f(X)$, resp. $f(X, Y)$, for each unary, resp. binary, symbol f in `BOOL-FVP` using the `get variants` command; see [9] for a theoretical justification of this check.

[6] Fresh variables follow the form `#1:Bool`.

`BOOL-FVP`. Indeed, $T_{\Sigma/E \cup B}$ is exactly the Booleans[7] on $\{$`true,false`$\}$ with the well-known truth tables for `and`, `xor`, `not`, `or` and `<=>`. This means that $E \cup B$-unification provides a Boolean *satisfiability decision procedure* for a Boolean expression u on such symbols, namely, u is Boolean satisfiable iff the equation $u =$ `true` is $E \cup B$-unifiable. Furthermore, a ground assignment ρ to the variables of u is a satisfying assignment for u iff there exists an $E \cup B$-unifier α of $u =$ `true` and a ground substitution δ such that $\rho = \alpha\delta$. For the same reasons, u is a Boolean *tautology* iff the equation $u =$ `false` has no $E \cup B$-unifiers.

A complete, finite set of $E \cup B$-unifiers can be computed with Maude's `variant unify` command whenever $E \cup B$ is FVP, except for the AwoC case. Instead, the new[8] `filtered variant unify` command computes a *finite, complete and minimal* set of $E \cup B$-unifiers, which can be considerably smaller than that computed by `variant unify`. For our `BOOL-FVP` example, `filtered variant unify` gives us a Boolean satisfiability decision procedure plus a symbolic specification of satisfying assignments. Such a procedure is not practical: it cannot compete with standard SAT-solvers; but that was never our purpose: our purpose here is to illustrate with simple examples how $E \cup B$-unification works for the *infinite* class of *user-definable* FVP theories $E \cup B$, of which `BOOL-FVP` is just a simple example; dozens of other examples can be found in [32].

The difference between the `variant unify` and the new `filtered variant unify` command is illustrated with the following example; its unfiltered output can be found in [10, §14]. Note that the single $E \cup B$-unifier gives us a compact symbolic description of this Boolean expression's satisfying assignments.

```
Maude> filtered variant unify (X or Y) <=> Z =? true .
rewrites: 3224 in 12765ms cpu (14776ms real) (252 rewrites/second)

Unifier 1
X --> #1:Bool xor #2:Bool
Y --> #1:Bool
Z --> #2:Bool xor (#1:Bool and (#1:Bool xor #2:Bool))

No more unifiers.
Advisory: Filtering was complete.
```

The computation of a minimal set of $E \cup B$-unifiers relies on filtering by $E \cup B$-matching between two $E \cup B$-unifiers, as explained in the following section.

3.1 FVP $E \cup B$-Matching and Minimality of $E \cup B$-Unifiers

By definition, a term u $E \cup B$*-matches* another term v iff there is a substitution γ such that $u =_{E \cup B} v\gamma$. Besides the existing `match` command modulo axioms

[7] Each connective's truth table can be checked with Maude's `reduce` command. Actually, need only check `and` and `xor` (other connectives are definitional extensions).

[8] In Maude, different command names are used to emphasize different algorithms. The word 'filtered' is used instead of 'irredundant' because irredundancy is not guaranteed in the AwoC case.

B, Maude's new `variant match` command computes a complete, minimal set of $E \cup B$-*matching substitutions* for any FVP theory $E \cup B$ in class (2), except for the AwoC case. Such an algorithm could always be derived from an $E \cup B$-unification algorithm by replacing u by $\overline{u}$, where all variables in u are replaced by fresh constants in $\overline{u}$, and computing the $E \cup B$-unifiers of $\overline{u} = v$. But a more efficient special-purpose algorithm has been designed and implemented for this purpose. $E \cup B$-matching algorithms are automatically provided by Maude for any *user-definable* theory in class (2) with the `variant match` command.

```
Maude> variant match in BOOL-FVP : Z and W <=? X .
rewrites: 12 in 21ms cpu (27ms real) (545 rewrites/second)

Matcher 1           Matcher 2           Matcher 3
Z --> true          Z --> X             Z --> X
W --> X             W --> true          W --> X
```

This is a good moment to ask and answer a relevant question: Why is computing a complete *minimal* set of $E \cup B$-unifiers for a unification problem Γ, where $E \cup B$ is an FVP theory in class (2) except for the AwoC case, *nontrivial*? We first need to explain how minimality is achieved. Suppose that α and β are two $E \cup B$-unifiers of a system of equations Γ with, say, typed variables $x_1, \ldots, x_n$. We then say that α *is more general than* β modulo $E \cup B$, denoted $\alpha \sqsupseteq_{E \cup B} \beta$, iff there is a substitution γ such that for each x_i, $1 \leq i \leq n$, $\gamma(\alpha(x_i)) =_{E \cup B} \beta(x_i)$. But this exactly means that the vector $[\beta(x_1), \ldots, \beta(x_n)]$ $E \cup B$-matches the vector $[\alpha(x_1), \ldots, \alpha(x_n)]$ with $E \cup B$-matching substitution γ. A complete set of $E \cup B$-unifiers of Γ is by definition *minimal* iff for any two different unifiers α and β in it we have $\alpha \not\sqsupseteq_{E \cup B} \beta$ *and* $\beta \not\sqsupseteq_{E \cup B} \alpha$, i.e., the two associated $E \cup B$-matching problems fail.

What is *nontrivial* is computing a minimal complete set of $E \cup B$-unifiers *efficiently*. One could do so inefficiently by simulating $E \cup B$-matching with $E \cup B$-unification, and more efficiently by using an $E \cup B$-matching algorithm. Maude achieves still greater efficiency by directly computing the $\alpha \sqsupseteq_{E \cup B} \beta$ relation. The key difference between the `variant unify` command and the new `filtered variant unify` command is that the second computes a $E \cup B$-minimal set of $E \cup B$-unifiers of Γ using the $\alpha \sqsupseteq_{E \cup B} \beta$ relation, whereas the first only computes a set of B-minimal $E \cup B$-unifiers of Γ using the cheaper $\alpha \sqsupseteq_{B} \beta$ relation. There are three ideas we use to make it fast in practice: (i) variant matching is faster than variant unification because one side is variable-free; (ii) enumerating the variant matchers between two variant unifiers is far more expensive than checking existence of a matcher; and (iii) variant unifiers are discarded on-the-fly avoiding further narrowing steps and computation.

4 Narrowing-Based Symbolic Reachability Analysis

In Maude, concurrent systems are specified in so-called *system modules* as *rewrite theories* of the form: $\mathcal{R} = (\Sigma, G, R)$, where G is an equational theory either of the form B in class (1), or $E \cup B$ in classes (2) or (3), and R are the *system transition rules*, specified as rewrite rules. When the theory $\mathcal{R}$ is *topmost*, meaning that the rules R rewrite the entire state, narrowing with rules R modulo the equations G is a *complete* symbolic reachability analysis method for *infinite-state systems* [35]. That is, given a term u with variables $\overrightarrow{x}$, representing a typically infinite set of initial states, and another term v with variables $\overrightarrow{y}$, representing a possibly infinite set of target states, narrowing can answer the question: *can an instance of* u *reach an instance of* v*?* That is, does the formula $\exists \overrightarrow{x}, \overrightarrow{y}\ \ u \to^* v$ hold in $\mathcal{R}$? Note that, if the *complement* of a system invariant I can be symbolically described as the set of ground instances of terms in a set $\{v_1, \dots, v_n\}$ of pattern terms, then narrowing provides a semi-decision procedure for verifying whether the system specified by $\mathcal{R}$ fails to satisfy I starting from an initial set of states specified by u. Namely, I holds iff no instance of any v_i can be reached from some instance of u.

Assuming G is in class (1) or (2), Maude's `vu-narrow` command implements narrowing with R modulo G by performing G-unification at each narrowing step. However, the number of symbolic states that need to be explored can be *infinite*. This means that if no solution exists for the narrowing search, Maude will search forever, so that only *depth-bounded searches* will terminate. The great advantage of the new `{fold} vu-narrow {filter,delay}` command is that it performs a powerful *symbolic state space reduction* by: (i) removing a newly explored symbolic state v' if it $E \cup B$-matches a previously explored state v and replacing transition with target v' by transitions with target v; and (ii) using minimal sets of $E \cup B$-unifiers for each narrowing step and for checking common instances between a newly explored state and the target term (ensured by words `filter` and `delay`). This can make the entire search space finite and allow full verification of invariants for some infinite-state systems. Consider the following Maude specification of Lamport's bakery protocol.

```
mod BAKERY is
  sorts Nat LNat Nat? State WProcs Procs .
  subsorts Nat LNat < Nat? .  subsort WProcs < Procs .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  op [_] : Nat -> LNat .            *** number-locking operator
  op < wait,_> : Nat -> WProcs .
  op < crit,_> : Nat -> Procs .
  op mt : -> WProcs .               *** empty multiset
  op __ : Procs Procs -> Procs [assoc comm id: mt] .    *** union
  op __ : WProcs WProcs -> WProcs [assoc comm id: mt] . *** union
  op _|_|_ : Nat Nat? Procs -> State .
  vars n m i j k : Nat . var x? : Nat? . var PS : Procs . var WPS : WProcs .

  rl [new]:  m | n | PS => s(m) | n | < wait,m > PS [narrowing] .
  rl [enter]: m | n | < wait,n > PS => m | [n] | < crit,n > PS [narrowing] .
  rl [leave]: m | [n] | < crit,n > PS => m | s(n) | PS [narrowing] .
endm
```

The states of `BAKERY` have the form "`m | x? | PS`" with `m` the ticket-dispensing counter, `x?` the (possibly locked) counter to access the critical section, and `PS` a multiset of processes either waiting or in the critical section. `BAKERY` is infinite-state: `[new]` creates new processes, and the counters can grow unboundedly. When a waiting process enters the critical section with `[enter]`, the second counter `n` is locked as `[n]`; and it is unlocked and incremented when it leaves it with `[leave]`. The key invariant is *mutual exclusion*. Note that the term "`i | x? | < crit, j > < crit, k > PS`" describes all states in the *complement* of mutual exclusion states. Without the `fold` option, narrowing does not terminate, but with the following command we can verify that `BAKERY` satisfies mutual exclusion, not just for the initial state "`0 | 0 | mt`", but for the much more general infinite set of initial states with waiting processes only "`m | n | WPS`".

```
Maude> {fold} vu-narrow {filter,delay}
        m | n | WPS =>* i | x? | < crit, j > < crit, k > PS .

No solution.
rewrites: 4 in 1ms cpu (1ms real) (2677 rewrites/second)
```

The new `vu-narrow {filter,delay}` command can achieve dramatic state space reductions over the previous `vu-narrow` command by filtering $E \cup B$-unifiers. This is illustrated by a simple cryptographic protocol example in [10, §15] exploiting the unitary nature of unification in the exclusive-or theory [24].

5 Applications and Conclusion

Maude can be used as a meta-tool to develop new formal tools because: (i) its underlying equational and rewriting logics are logical —and reflective meta-logical— frameworks [7,27,46]; (ii) Maude's efficient support of logical reflection through its `META-LEVEL` module; (iii) Maude's rewriting, search, model checking, and strategy language features [11,15]; and (iv) Maude's symbolic reasoning features [15,33], the latest reported here. We refer to [11,15,31,33] for references on various Maude-based tools. Many of them can benefit from these new features.

By way of example we mention some areas ready to reap such benefits: (1) *Formal Analysis of Cryptographic Protocols.* The new features can yield substantial improvements to tools such as Maude-NPA [19], Tamarin [28] and AKISS [8]. (2) *Model Checking of Infinite-State Systems.* The narrowing-based LTL symbolic model checker reported in [6,20], and the addition of new symbolic capabilities to Real-Time Maude [37,38] can both benefit from the new features. (3) *SMT Solving.* In Sect. 3 we noted that FVP $E \cup B$-unification makes satisfiability of positive QF formulas in $T_{\Sigma/E\cup B}$ decidable. Under mild conditions, this has been extended in [32,44] to a procedure for satisfiability in $T_{\Sigma/E\cup B}$ of all QF formulas which will also benefit from the new features. (4) *Theorem Proving.* The new Maude Inductive Theorem Prover under construction [34], as well as

Maude's Invariant Analyzer [43] and Reachability Logic Theorem Prover [45] all use equational unification and narrowing modulo equations; so all will benefit from the new features. (5) *Theory Transformations* based on equational unification, e.g., partial evaluation [4], ground confluence methods [17] or program termination methods [25, 26] could likewise become more efficient.

In conclusion, we have presented and illustrated with examples new equational unification and matching, and symbolic reachability analysis features in Maude 3.2. Thanks to the above-mentioned properties (i)–(iv) of Maude as a meta-tool, we hope that this work will encourage other researchers to use Maude and its symbolic features to develop new tools in many different logics.

References

1. Aït-Kaci, H., Sasaki, Y.: An axiomatic approach to feature term generalization. In: De Raedt, L., Flach, P. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, pp. 1–12. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44795-4_1
2. Alpuente, M., Ballis, D., Cuenca-Ortega, A., Escobar, S., Meseguer, J.: ACUOS^2: a high-performance system for modular ACU generalization with subtyping and inheritance. In: Calimeri, F., Leone, N., Manna, M. (eds.) JELIA 2019. LNCS (LNAI), vol. 11468, pp. 171–181. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19570-0_11
3. Alpuente, M., Cuenca-Ortega, A., Escobar, S., Meseguer, J.: Order-sorted homeomorphic embedding modulo combinations of associativity and/or commutativity axioms. Fundam. Inform. **177**(3–4), 297–329 (2020)
4. Alpuente, M., Cuenca-Ortega, A., Escobar, S., Meseguer, J.: A partial evaluation framework for order-sorted equational programs modulo axioms. J. Log. Algebraic Methods Program. **110**, 100501 (2020)
5. Alpuente, M., Falaschi, M., Vidal, G.: Partial evaluation of functional logic programs. ACM Trans. Program. Lang. Syst. **20**(4), 768–844 (1998)
6. Bae, K., Escobar, S., Meseguer, J.: Abstract logical model checking of infinite-state systems using narrowing. In: RTA 2013. LIPIcs, vol. 21, pp. 81–96. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013)
7. Basin, D., Clavel, M., Meseguer, J.: Rewriting logic as a metalogical framework. ACM Trans. Comput. Log. **5**, 528–576 (2004)
8. Chadha, R., Cheval, V., Ciobâcă, Ş, Kremer, S.: Automated verification of equivalence properties of cryptographic protocols. ACM Trans. Comput. Log. **17**(4), 23:1–23:32 (2016)
9. Cholewa, A., Meseguer, J., Escobar, S.: Variants of variants and the finite variant property. Technical report, CS Dept. University of Illinois at Urbana-Champaign, February 2014. http://hdl.handle.net/2142/47117
10. Clavel, M., et al.: Maude manual (version 3.2.1). SRI International, February 2022. http://maude.cs.illinois.edu
11. Clavel, M., et al.: All About Maude, A High-Performance Logical Framework. Lecture Notes in Computer Science, vol. 4350. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71999-1
12. Colmerauer, A.: An introduction to Prolog III. Commun. ACM **33**(7), 69–90 (1990)
13. Comon-Lundh, H., Delaune, S.: The finite variant property: how to get rid of some algebraic properties. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 294–307. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-32033-3_22

14. Dershowitz, N., Jouannaud, J.-P.: Rewrite systems. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, pp. 243–320. North-Holland (1990)
15. Durán, F., et al.: Programming and symbolic computation in Maude. J. Log. Algebraic Methods Program. **110**, 100497 (2020)
16. Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Talcott, C.: Associative unification and symbolic reasoning modulo associativity in Maude. In: Rusu, V. (ed.) WRLA 2018. LNCS, vol. 11152, pp. 98–114. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99840-4_6
17. Durán, F., Meseguer, J., Rocha, C.: Ground confluence of order-sorted conditional specifications modulo axioms. J. Log. Algebraic Methods Program. **111**, 100513 (2020)
18. Eker, S.: Associative unification in Maude. J. Log. Algebraic Methods Program. **126**, 100747 (2022)
19. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: cryptographic protocol analysis modulo equational properties. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) FOSAD 2007-2009. LNCS, vol. 5705, pp. 1–50. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03829-7_1
20. Escobar, S., Meseguer, J.: Symbolic model checking of infinite-state systems using narrowing. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 153–168. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73449-9_13
21. Escobar, S., Sasse, R., Meseguer, J.: Folding variant narrowing and optimal variant termination. J. Algebraic Log. Program. **81**, 898–928 (2012)
22. Goguen, J., Meseguer, J.: Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theoret. Comput. Sci. **105**, 217–273 (1992)
23. Jaffar, J., Maher, M.J.: Constraint logic programming: a survey. J. Log. Program. **19**(20), 503–581 (1994)
24. Kapur, D., Narendran, P.: Matching, unification and complexity. SIGSAM Bull. **21**(4), 6–9 (1987)
25. Lucas, S., Meseguer, J., Gutiérrez, R.: The 2D dependency pair framework for conditional rewrite systems. Part I: definition and basic processors. J. Comput. Syst. Sci. **96**, 74–106 (2018)
26. Lucas, S., Meseguer, J., Gutiérrez, R.: The 2D dependency pair framework for conditional rewrite systems - Part II: advanced processors and implementation techniques. J. Autom. Reason. **64**(8), 1611–1662 (2020)
27. Martí-Oliet, N., Meseguer, J.: Rewriting logic as a logical and semantic framework. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, 2nd. Edition, pages 1–87. Kluwer Academic Publishers (2002). First published as SRI Technical report SRI-CSL-93-05, August 1993
28. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 696–701. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_48
29. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Presicce, F.P. (ed.) WADT 1997. LNCS, vol. 1376, pp. 18–61. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-64299-4_26
30. Meseguer, J.: Strict coherence of conditional rewriting modulo axioms. Theor. Comput. Sci. **672**, 1–35 (2017)

31. Meseguer, J.: Symbolic reasoning methods in rewriting logic and Maude. In: Moss, L.S., de Queiroz, R., Martinez, M. (eds.) WoLLIC 2018. LNCS, vol. 10944, pp. 25–60. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-57669-4_2
32. Meseguer, J.: Variant-based satisfiability in initial algebras. Sci. Comput. Program. **154**, 3–41 (2018)
33. Meseguer, J.: Symbolic computation in Maude: some tapas. In: LOPSTR 2020. LNCS, vol. 12561, pp. 3–36. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-68446-4_1
34. Meseguer, J., Skeirik, S.: Inductive reasoning with equality predicates, contextual rewriting and variant-based simplification. In: Escobar, S., Martí-Oliet, N. (eds.) WRLA 2020. LNCS, vol. 12328, pp. 114–135. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63595-4_7
35. Meseguer, J., Thati, P.: Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. High.-Order Symb. Comput. **20**(1–2), 123–160 (2007)
36. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 371–443. Elsevier and MIT Press (2001)
37. Ölveczky, P.C.: Real-time Maude and its applications. In: Escobar, S. (ed.) WRLA 2014. LNCS, vol. 8663, pp. 42–79. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12904-4_3
38. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of real-time Maude. High.-Order Symb. Comput. **20**(1–2), 161–196 (2007)
39. Peterson, G.E., Stickel, M.E.: Complete sets of reductions for some equational theories. J. Assoc. Comput. Mach. **28**(2), 233–264 (1981)
40. Plotkin, G.: Building-in equational theories. In: Meltzer, B., Michie, D. (eds.) 1971 Proceedings of the Seventh Annual Machine Intelligence Workshop on Machine Intelligence 7, Edinburgh, pp. 73–90. Edinburgh University Press (1972)
41. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. Assoc. Comput. Mach. **12**(1), 23–41 (1965)
42. Rocha, C., Meseguer, J.: Five isomorphic Boolean theories and four equational decision procedures. Technical report UIUCDCS-R-2007-2818, CS Department, University of Illinois at Urbana-Champaign, February 2007. http://hdl.handle.net/2142/11295
43. Rocha, C., Meseguer, J.: Proving safety properties of rewrite theories. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 314–328. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22944-2_22
44. Skeirik, S., Meseguer, J.: Metalevel algorithms for variant satisfiability. J. Log. Algebr. Meth. Program. **96**, 81–110 (2018)
45. Skeirik, S., Stefanescu, A., Meseguer, J.: A constructor-based reachability logic for rewrite theories. Fundam. Inform. **173**(4), 315–382 (2020)
46. Stehr, M.-O., Meseguer, J.: Pure type systems in rewriting logic: specifying typed higher-order languages in a first-order logical framework. In: Owe, O., Krogdahl, S., Lyche, T. (eds.) From Object-Orientation to Formal Methods. LNCS, vol. 2635, pp. 334–375. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-39993-3_16
47. Walther, C.: A mechanical solution of Schubert's steamroller by many-sorted resolution. Artif. Intell. **26**(2), 217–224 (1985)
48. Zheng, Y., et al.: Z3str2: an efficient solver for strings, regular expressions, and length constraints. Formal Methods Syst. Design **50**(2–3), 249–288 (2017)

Leśniewski's Ontology – Proof-Theoretic Characterization

Andrzej Indrzejczak[✉]

Department of Logic, University of Lodz, Łódź, Poland
andrzej.indrzejczak@filhist.uni.lodz.pl

Abstract. The ontology of Leśniewski is commonly regarded as the most comprehensive calculus of names and the theoretical basis of mereology. However, ontology was not examined by means of proof-theoretic methods so far. In the paper we provide a characterization of elementary ontology as a sequent calculus satisfying desiderata usually formulated for rules in well-behaved systems in modern structural proof theory. In particular, the cut elimination theorem is proved and the version of subformula property holds for the cut-free version.

Keywords: Leśniewski · Ontology · Calculus of Names · Sequent Calculus · Cut Elimination

1 Introduction

The ontology of Leśniewski is a kind of calculus of names proposed as a formalization of logic alternative to Fregean paradigm. Basically, it is a theory of the binary predicate ε understood as the formalization of the Greek 'esti'. Informally a formula $a\varepsilon b$ is to be read as "(the) a is (a/the) b", so in order to be true a must be an individual name whereas b can be individual or general name. In the original formulation Leśniewski's ontology is the middle part of the hierarchical structure involving also the protothetics and mereology (see the presentation in Urbaniak [20]). Protothetics, a very general form of propositional logic, is the basis of the overall construction. Its generality follows from the fact that, in addition to sentence variables, arbitrary sentence-functors (connectives) are allowed as variables, and quantifiers binding all these kinds of variables are involved. Similarly in Leśniewski's ontology, we have a quantification over name variables but also over arbitrary name-functors creating complex names. In consequence we obtain very expressive logic which is then extended to mereology. The latter, which is the most well-known ingredient of Leśniewski's construction, is a theory of parthood relation, which provides an alternative formalization of the theory of classes and foundations of mathematics.

Despite of the dependence of Leśniewski's ontology on his protothetics, we can examine this theory, in particular its part called elementary ontology, in isolation, as a kind of first-order theory of ε based on classical first-order logic (FOL). Elementary ontology, in this sense, was investigated, among others, by

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 541–558, 2022.
https://doi.org/10.1007/978-3-031-10769-6_32

Słupecki [17] and Iwanuś [7], and we follow this line here. The expressive power of such an approach is strongly reduced, in particular, quantifiers apply only to name variables. One should note however that, despite of the appearances, it is not just another elementary theory in the standard sense, since the range of variables is not limited to individual names but admits general and even empty names. Thus, name variables may represent not only 'Napoleon Bonaparte' but also 'an emperor' and 'Pegasus'. This leads to several problems concerning the interpretation of quantifiers in ontology, encountered in the semantical treatment (see e.g. Küng and Canty [8] or Rickey [16]). However, for us the problems of proper interpretation are not important here, since we develop purely syntactical formulation, which is shown to be equivalent to Leśniewski's axiomatic formulation.

Taking into account the importance and originality of Leśniewski's ontology it is interesting, if not surprising, that so far no proof-theoretic study was offered, in particular, in terms of sequent calculus (SC). In fact, a form of natural deduction proof system was applied by many authors following the original way of presenting proofs by Leśniewski (see, e.g. his [9–11]). However this can hardly be treated as a proof-theoretic study of Leśniewski's ontology but only as a convenient way of simplifying presentation of axiomatic proofs. Ishimoto and Kobayashi [6] introduced also a tableau system for part of (quantifier-free) ontology – we will say more about this system later.

In this paper we present a sequent calculus for elementary ontology and focus on its most important properties. More specifically, in Sect. 2 we briefly characterise elementary ontology which will be the object of our study. In Sect. 3 we present an adequate sequent calculus for the basic part of elementary ontology and prove that it is equivalent with the axiomatic formulation. Then we prove the cut elimination theorem for this calculus in Sect. 4. In the next section we focus on the problem of extensionality and discuss some alternative formulations of ontology and some of its parts, as well as the intuitionistic version of it. Section 6 shows how the basic system can be extended with rules for new predicate constants which preserve cut elimination. The problem of extension with rules for term constants is discussed briefly in Sect. 7. A summary of obtained results and open problems closes the paper.

2 Elementary Ontology

Roughly, in this article, by Leśniewski's elementary ontology we mean standard FOL (in some chosen adequate formalization) with Leśniewski's axiom LA added. For more detailed general presentation of Leśniewski's systems one may consult Urbaniak [20] and for a detailed study of Leśniewski's ontology see Iwanuś [7] or Słupecki [17]. In the next section we will select a particular sequent system as representing FOL and investigate several ways of possible representation of LA in this framework.

We will consider two languages for ontology. In both we assume a denumerable set of name variables. Following the well-known Gentzen's custom we apply a graphical distinction between the bound variables, which will be denoted by $x, y, z, ...$ (possibly with subscripts), and the free variables usually called parameters, which will be denoted by $a, b, c, ...$ These are the only terms we admit, and both kinds will be called simply name variables. The basic language L_o consists of the following vocabulary:

- connectives: $\neg, \wedge, \vee, \rightarrow$;
- first-order quantifiers: $\forall, \exists$;
- predicate: ε.

As we can see, in addition to the standard logical vocabulary of FOL, the only specific constant is a binary predicate ε with the formation rule: $t \varepsilon t'$ is an atomic formula, for any terms t, t'. In what follows we will use a convention: instead of $t \varepsilon t'$ we will write tt'. The complexity of formulae of L_o is defined as the number of occurrences of logical constants, i.e. connectives and quantifiers. Hence the complexity of atomic formulae is 0.

The language L_p, considered in Sect. 6, adds to this vocabulary a number of unary and binary predicates: $D, V, S, G, U, =, \equiv, \approx, \bar{\varepsilon}, \subset, \not\subseteq, A, E, I, O$.

In L_o and L_p we have name variables, which range over all names (individual, general and empty), as the only terms. However Leśniewski considered also complex terms built with the help of specific term-forming functors. We will discuss briefly such extensions in the setting of sequent calculus in Sect. 7 and notice important problems they generate for decent proof-theoretic treatment.

The only specific axiom of elementary ontology is Leśniewski's axiom LA:

$$\forall xy(xy \leftrightarrow \exists z(zx) \wedge \forall z(zx \rightarrow zy) \wedge \forall zv(zx \wedge vx \rightarrow zv))$$

$\text{LA}^{\rightarrow}$, $\text{LA}^{\leftarrow}$ will be used to refer to the respective implications forming LA, with dropped outer universal quantifier. Note that:

Lemma 1. *The following formulae are equivalent to LA:*

1. $\forall xy(xy \leftrightarrow \exists z(zx \wedge zy) \wedge \forall zv(zx \wedge vx \rightarrow zv))$
2. $\forall xy(xy \leftrightarrow \exists z(zx \wedge zy \wedge \forall v(vx \rightarrow vz)))$
3. $\forall xy(xy \leftrightarrow \exists z(\forall v(vx \leftrightarrow vz) \wedge zy))$

We start with the system in the language L_o, i.e. with ε (conventionally omitted) as the only specific predicate constant added to the standard language of FOL.

3 Sequent Calculus

Elementary ontology will be formalised as a sequent calculus with sequents $\Gamma \Rightarrow \Delta$ which are ordered pairs of finite multisets of formulae called the antecedent and the succedent, respectively. We will use the calculus G (after Gentzen) which is essentially the calculus G1 of Troelstra and Schwichtenberg [19]. All necessary

$$(AX)\ \varphi \Rightarrow \varphi \qquad (Cut)\ \frac{\Gamma \Rightarrow \Delta, \varphi \qquad \varphi, \Pi \Rightarrow \Sigma}{\Gamma, \Pi \Rightarrow \Delta, \Sigma}$$

$$(W\!\Rightarrow)\ \frac{\Gamma \Rightarrow \Delta}{\varphi, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow\! W)\ \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \varphi}$$

$$(C\!\Rightarrow)\ \frac{\varphi, \varphi, \Gamma \Rightarrow \Delta}{\varphi, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow\! C)\ \frac{\Gamma \Rightarrow \Delta, \varphi, \varphi}{\Gamma \Rightarrow \Delta, \varphi}$$

$$(\neg\!\Rightarrow)\ \frac{\Gamma \Rightarrow \Delta, \varphi}{\neg\varphi, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow\!\neg)\ \frac{\varphi, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg\varphi}$$

$$(\wedge\!\Rightarrow)\ \frac{\varphi, \psi, \Gamma \Rightarrow \Delta}{\varphi \wedge \psi, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow\!\wedge)\ \frac{\Gamma \Rightarrow \Delta, \varphi \qquad \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \wedge \psi}$$

$$(\vee\!\Rightarrow)\ \frac{\varphi, \Gamma \Rightarrow \Delta \qquad \psi, \Gamma \Rightarrow \Delta}{\varphi \vee \psi, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow\!\vee)\ \frac{\Gamma \Rightarrow \Delta, \varphi, \psi}{\Gamma \Rightarrow \Delta, \varphi \vee \psi}$$

$$(\rightarrow\Rightarrow)\ \frac{\Gamma \Rightarrow \Delta, \varphi \qquad \psi, \Gamma \Rightarrow \Delta}{\varphi \rightarrow \psi, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow\rightarrow)\ \frac{\varphi, \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \rightarrow \psi}$$

$$(\leftrightarrow\Rightarrow)\ \frac{\Gamma \Rightarrow \Delta, \varphi, \psi \qquad \varphi, \psi, \Gamma \Rightarrow \Delta}{\varphi \leftrightarrow \psi, \Gamma \Rightarrow \Delta} \qquad (\forall\!\Rightarrow)\ \frac{\varphi[x/b], \Gamma \Rightarrow \Delta}{\forall x \varphi, \Gamma \Rightarrow \Delta}$$

$$(\Rightarrow\leftrightarrow)\ \frac{\varphi, \Gamma \Rightarrow \Delta, \psi \qquad \psi, \Gamma \Rightarrow \Delta, \varphi}{\Gamma \Rightarrow \Delta, \varphi \leftrightarrow \psi} \qquad (\Rightarrow\!\forall)\ \frac{\Gamma \Rightarrow \Delta, \varphi[x/a]}{\Gamma \Rightarrow \Delta, \forall x \varphi}$$

$$(\exists\!\Rightarrow)\ \frac{\varphi[x/a], \Gamma \Rightarrow \Delta}{\exists x \varphi, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow\!\exists)\ \frac{\Gamma \Rightarrow \Delta, \varphi[x/b]}{\Gamma \Rightarrow \Delta, \exists x \varphi}$$

where a is a fresh parameter (eigenvariable), not present in Γ, Δ and φ, whereas b is an arbitrary parameter.

Fig. 1. Calculus G

structural rules, including cut, weakening and contraction are primitive. The calculus G consists of the rules from Fig. 1:

Let us recall that formulae displayed in the schemata are active, whereas the remaining ones are parametric, or form a context. In particular, all active formulae in the premisses are called side formulae, and the one in the conclusion is the principal formula of the respective rule application. Proofs are defined in a standard way as finite trees with nodes labelled by sequents. The height of a proof $\mathcal{D}$ of $\Gamma \Rightarrow \Delta$ is defined as the number of nodes of the longest branch in $\mathcal{D}$. $\vdash_k \Gamma \Rightarrow \Delta$ means that $\Gamma \Rightarrow \Delta$ has a proof of the height at most k.

G provides an adequate formalization of the classical pure FOL (i.e. with no terms other than variables). However, we should remember that here terms in quantifier rules are restricted to variables ranging over arbitrary names (including empty and general). This means, in particular, that quantifiers do not have an existential import, like in standard FOL.

Let us call G+LA an extension of G with LA as an additional axiomatic sequent. The following hold:

Lemma 2. *The following sequents are provable in G+LA:*
$ab \Rightarrow \exists x(xa)$
$ab \Rightarrow \forall x(xa \to xb)$
$ab \Rightarrow \forall xy(xa \wedge ya \to xy)$
$\exists x(xa), \forall x(xa \to xb), \forall xy(xa \wedge ya \to xy) \Rightarrow ab$

The proof is obvious. In fact, these sequents together allow us to derive LA so we could use them alternatively in a characterization of elementary ontology on the basis of G.

G+LA is certainly an adequate formalization of elementary ontology in the sense of Słupecki and Iwanuś. However, from the standpoint of proof theoretic analysis it is not an interesting form of sequent calculus and it will be used only for showing the adequacy of our main system called GO.

To obtain the basic GO we add the following four rules to G:

$$(R)\ \dfrac{aa, \Gamma \Rightarrow \Delta}{ab, \Gamma \Rightarrow \Delta} \qquad (T)\ \dfrac{ac, \Gamma \Rightarrow \Delta}{ab, bc, \Gamma \Rightarrow \Delta} \qquad (S)\ \dfrac{ba, \Gamma \Rightarrow \Delta}{ab, bb, \Gamma \Rightarrow \Delta}$$

$$(E)\ \dfrac{da, \Gamma \Rightarrow \Delta, dc \quad dc, \Gamma \Rightarrow \Delta, da \quad ab, \Gamma \Rightarrow \Delta}{cb, \Gamma \Rightarrow \Delta}$$

where d in (E) is a new parameter (eigenvariable), and a, b, c are arbitrary.

The names of rules come from reflexivity, transitivity, symmetry and extensionality. In case of (R) and (S) it is a kind of prefixed reflexivity and symmetry $(ab \to aa, bb \to (ab \to ba))$. Why (E) comes from extensionality will be explained later.

We can show that GO is an adequate characterization of elementary ontology.

Theorem 1. *If G+LA $\vdash \Gamma \Rightarrow \Delta$, then GO $\vdash \Gamma \Rightarrow \Delta$.*

Proof. It is sufficient to prove that the axiomatic sequent LA is provable in GO.

$$\dfrac{(\Rightarrow \exists)\ \dfrac{(R)\ \dfrac{aa \Rightarrow aa}{ab \Rightarrow aa}}{ab \Rightarrow \exists x(xa)} \qquad \dfrac{\dfrac{\dfrac{cb \Rightarrow cb}{ca, ab \Rightarrow cb}\ (T)}{ab \Rightarrow ca \to cb}\ (\Rightarrow\to)}{ab \Rightarrow \forall x(xa \to xb)}\ (\Rightarrow \forall)}{ab \Rightarrow \exists x(xa) \wedge \forall x(xa \to xb)}\ (\Rightarrow \wedge)$$

$(\Rightarrow \wedge)$ with:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{cd \Rightarrow cd}{ca, ad \Rightarrow cd}\ (T)}{ca, da, aa \Rightarrow cd}\ (S)}{ca, da, ab \Rightarrow cd}\ (R)}{ab, ca \wedge da \Rightarrow cd}\ (\wedge \Rightarrow)}{ab \Rightarrow ca \wedge da \to cd}\ (\Rightarrow\to)}{ab \Rightarrow \forall xy(xa \wedge ya \to xy)}\ (\Rightarrow \forall)$$

yields LA$^{\to}$ after $(\Rightarrow\to)$. A proof of the converse is more complicated (for readability and space-saving we ommited all applications of weakening rules necessary for the application of two- and three-premiss rules; this convention will be applied hereafter with no comments):

$$\dfrac{\dfrac{ca \Rightarrow ca \qquad (E)\,\dfrac{(\forall\Rightarrow)\,\dfrac{(\to\Rightarrow)\,\dfrac{(\Rightarrow\wedge)\,\dfrac{da \Rightarrow da \qquad ca \Rightarrow ca}{da, ca \Rightarrow da \wedge ca} \qquad dc \Rightarrow dc}{da, ca, da \wedge ca \to dc \Rightarrow dc}}{da, ca, \forall xy(xa \wedge ya \to xy) \Rightarrow dc} \qquad \dfrac{da \Rightarrow da}{dc, ca \Rightarrow da}\,(T) \qquad ab \Rightarrow ab}{cb, ca, \forall xy(xa \wedge ya \to xy) \Rightarrow ab}}{ca, ca \to cb, \forall xy(xa \wedge ya \to xy) \Rightarrow ab}\,(\to\Rightarrow)}{\dfrac{ca, \forall x(xa \to xb), \forall xy(xa \wedge ya \to xy) \Rightarrow ab}{\exists x(xa), \forall x(xa \to xb), \forall xy(xa \wedge ya \to xy) \Rightarrow ab}\,(\exists\Rightarrow)}\,(\forall\Rightarrow)$$

It is routine to prove LA. □

Note that to prove LA$^{\to}$ the rules $(R), (T), (S)$ were sufficient, whereas in order to derive the converse, (E) alone is not sufficient - we need (T) again.

Theorem 2. *If GO $\vdash \Gamma \Rightarrow \Delta$, then G+LA $\vdash \Gamma \Rightarrow \Delta$.*

Proof. It is sufficient to prove that the four rules of GO are derivable in G+LA.

For (T):

$$\dfrac{\dfrac{bc \Rightarrow \forall x(xb \to xc) \qquad \dfrac{\dfrac{ab \Rightarrow ab \qquad ac \Rightarrow ac}{ab \to ac, ab \Rightarrow ac}\,(\to\Rightarrow)}{\forall x(xb \to xc), ab \Rightarrow ac}\,(\forall\Rightarrow)}{ab, bc \Rightarrow ac}\,(Cut) \qquad ac, \Gamma \Rightarrow \Delta}{ab, bc, \Gamma \Rightarrow \Delta}\,(Cut)$$

where the leftmost leaf is provable in G+LA (Lemma 2).

For (S):

$$\dfrac{\dfrac{bb \Rightarrow \forall xy(xb \wedge yb \to xy) \qquad \dfrac{\dfrac{\dfrac{bb \Rightarrow bb \qquad ab \Rightarrow ab}{bb, ab \Rightarrow bb \wedge ab}\,(\Rightarrow\wedge) \qquad ba \Rightarrow ba}{bb \wedge ab \to ba, bb, ab \Rightarrow ba}\,(\to\Rightarrow)}{\forall xy(xb \wedge yb \to xy), bb, ab \Rightarrow ba}\,(\forall\Rightarrow)}{bb, bb, ab \Rightarrow ba}\,(Cut)}{bb, ab \Rightarrow ba}\,(C\Rightarrow)$$

where the leftmost leaf is provable in G+LA (Lemma 2). By cut with the premiss of (S) we obtain its conclusion.

For (R):

$$\dfrac{\dfrac{ab \Rightarrow \forall xy(xa \wedge ya \to xy) \qquad \dfrac{ab \Rightarrow \exists x(xa) \qquad S}{\forall xy(xa \wedge ya \to xy), \forall x(xa \to xa), ab \Rightarrow aa}\,(Cut)}{\forall x(xa \to xa), ab, ab \Rightarrow aa}\,(Cut)}{\forall x(xa \to xa), ab \Rightarrow aa}\,(C\Rightarrow)$$

where $S := \exists x(xa), \forall xy(xa \wedge ya \rightarrow xy), \forall x(xa \rightarrow xa) \Rightarrow aa$ and all leaves are provable in G+LA (Lemma 2); in particular S is the fourth sequent with b replaced with a. By cut with $\Rightarrow \forall x(xa \rightarrow xa)$ and the premiss of (R) we obtain its conclusion.

Since $(R), (T), (S)$ are all derivable in G+LA we use them in the proof of the derivability of (E) to simplify matters. Note first the following three proofs with weakenings omitted:

$$\begin{prooftree}
\AxiomC{$cc \Rightarrow cc$}
\LeftLabel{(R)}
\UnaryInfC{$cb \Rightarrow cc$}
\AxiomC{$ca \Rightarrow ca$}
\LeftLabel{$(\leftrightarrow\Rightarrow)$}
\BinaryInfC{$ca \leftrightarrow cc, cb \Rightarrow ca$}
\LeftLabel{$(\Rightarrow \exists)$}
\UnaryInfC{$ca \leftrightarrow cc, cb \Rightarrow \exists x(xa)$}
\LeftLabel{$(\forall \Rightarrow)$}
\UnaryInfC{$\forall x(xa \leftrightarrow xc), cb \Rightarrow \exists x(xa)$}
\end{prooftree}$$

$$\begin{prooftree}
\AxiomC{$da \Rightarrow da$}
\AxiomC{$db \Rightarrow db$}
\RightLabel{(T)}
\UnaryInfC{$dc, cb \Rightarrow db$}
\LeftLabel{$(\leftrightarrow\Rightarrow)$}
\BinaryInfC{$da \leftrightarrow dc, cb, da \Rightarrow db$}
\LeftLabel{$(\forall \Rightarrow)$}
\UnaryInfC{$\forall x(xa \leftrightarrow xc), cb, da \Rightarrow db$}
\LeftLabel{$(\Rightarrow\rightarrow)$}
\UnaryInfC{$\forall x(xa \leftrightarrow xc), cb \Rightarrow da \rightarrow db$}
\LeftLabel{$(\Rightarrow \forall)$}
\UnaryInfC{$\forall x(xa \leftrightarrow xc), cb \Rightarrow \forall x(xa \rightarrow xb)$}
\end{prooftree}$$

and

$$\begin{prooftree}
\AxiomC{$da \Rightarrow da$}
\AxiomC{$ea \Rightarrow ea$}
\AxiomC{$de \Rightarrow de$}
\RightLabel{(T)}
\UnaryInfC{$ce, dc \Rightarrow de$}
\RightLabel{(S)}
\UnaryInfC{$ec, dc, cc \Rightarrow de$}
\RightLabel{$(R \Rightarrow)$}
\UnaryInfC{$ec, dc, cb \Rightarrow de$}
\RightLabel{$(\leftrightarrow\Rightarrow)$}
\BinaryInfC{$dc, ea \leftrightarrow ec, cb, ea \Rightarrow de$}
\RightLabel{$(\forall \Rightarrow)$}
\UnaryInfC{$dc, \forall x(xa \leftrightarrow xc), cb, ea \Rightarrow de$}
\LeftLabel{$(\leftrightarrow\Rightarrow)$}
\BinaryInfC{$da \leftrightarrow dc, \forall x(xa \leftrightarrow xc), cb, da, ea \Rightarrow de$}
\LeftLabel{$(\forall \Rightarrow)$}
\UnaryInfC{$\forall x(xa \leftrightarrow xc), \forall x(xa \leftrightarrow xc), cb, da, ea \Rightarrow de$}
\LeftLabel{$(C \Rightarrow)$}
\UnaryInfC{$\forall x(xa \leftrightarrow xc), cb, da, ea \Rightarrow de$}
\LeftLabel{$(\wedge \Rightarrow)$}
\UnaryInfC{$\forall x(xa \leftrightarrow xc), cb, da \wedge ea \Rightarrow de$}
\LeftLabel{$(\Rightarrow\rightarrow)$}
\UnaryInfC{$\forall x(xa \leftrightarrow xc), cb \Rightarrow da \wedge ea \rightarrow de$}
\LeftLabel{$(\Rightarrow \forall)$}
\UnaryInfC{$\forall x(xa \leftrightarrow xc), cb \Rightarrow \forall xy(xa \wedge ya \rightarrow xy)$}
\end{prooftree}$$

By three cuts with $\exists x(xa), \forall x(xa \rightarrow xb), \forall xy(xa \wedge ya \rightarrow xy) \Rightarrow ab$ and contractions we obtain a proof of $S := \forall x(xa \leftrightarrow xc), cb \Rightarrow ab$. Then we finish in the following way:

$$\begin{prooftree}
\AxiomC{$da, \Gamma \Rightarrow \Delta, dc$}
\AxiomC{$dc, \Gamma \Rightarrow \Delta, da$}
\LeftLabel{$(\Rightarrow\leftrightarrow)$}
\BinaryInfC{$\Gamma \Rightarrow \Delta, da \leftrightarrow dc$}
\LeftLabel{$(\Rightarrow \forall)$}
\UnaryInfC{$\Gamma \Rightarrow \Delta, \forall x(xa \leftrightarrow xc)$}
\AxiomC{S}
\LeftLabel{(Cut)}
\BinaryInfC{$cb, \Gamma \Rightarrow \Delta, ab$}
\AxiomC{$ab, \Gamma \Rightarrow \Delta$}
\LeftLabel{(Cut)}
\BinaryInfC{$cb, \Gamma \Rightarrow \Delta$}
\end{prooftree}$$

Note that to prove derivability of (E) we need in fact the whole LA. We elaborate on the strength of this rule in Sect. 5. □

4 Cut Elimination

The possibility of representing LA by means of these four rules makes GO a calculus with desirable proof-theoretic properties. First of all note that for G the cut elimination theorem holds. Since the only primitive rules for ε are all one-sided, in the sense that principal formulae occur in the antecedents only, we can easily extend this result to GO. We follow the general strategy of cut elimination proofs applied originally for hypersequent calculi in Metcalfe, Olivetti and Gabbay [13] but which works well also in the context of standard sequent calculi (see Indrzejczak [5]). Such a proof has a particularly simple structure and allows us to avoid many complexities inherent in other methods of proving cut elimination. In particular, we avoid well known problems with contraction, since two auxiliary lemmata deal with this problem in advance. Note first that for GO the following result holds:

Lemma 3 (Substitution). *If $\vdash_k \Gamma \Rightarrow \Delta$, then $\vdash_k \Gamma[a/b] \Rightarrow \Delta[a/b]$.*

Proof. By induction on the height of a proof. Note that (E) may require similar relettering like $(\exists\Rightarrow)$ and $(\Rightarrow\forall)$. Note that the proof provides the height-preserving admissibility of substitution. □

Let us assume that all proofs are regular in the sense that every parameter a which is fresh by side condition on the respective rule must be fresh in the entire proof, not only on the branch where the application of this rule takes place. There is no loss of generality since every proof may be systematically transformed into a regular one by the substitution lemma. The following notions are crucial for the proof:

1. The cut-degree is the complexity of cut-formula φ, i.e. the number of connectives and quantifiers occurring in φ; it is denoted as $d\varphi$.
2. The proof-degree ($d\mathcal{D}$) is the maximal cut-degree in $\mathcal{D}$.

Remember that the complexity of atomic formulae, and consequently of cut- and proof-degree in case of atomic cuts, is 0. The proof of the cut elimination theorem is based on two lemmata which successively make a reduction: first on the height of the right, and then on the height of the left premiss of cut. φ^k, Γ^k denote $k > 0$ occurrences of φ, Γ, respectively.

Lemma 4 (Right reduction). *Let $\mathcal{D}_1 \vdash \Gamma \Rightarrow \Delta, \varphi$ and $\mathcal{D}_2 \vdash \varphi^k, \Pi \Rightarrow \Sigma$ with $d\mathcal{D}_1, d\mathcal{D}_2 < d\varphi$, and φ principal in $\Gamma \Rightarrow \Delta, \varphi$, then we can construct a proof $\mathcal{D}$ such that $\mathcal{D} \vdash \Gamma^k, \Pi \Rightarrow \Delta^k, \Sigma$ and $d\mathcal{D} < d\varphi$.*

Proof. By induction on the height of $\mathcal{D}_2$. The basis is trivial, since $\Gamma \Rightarrow \Delta, \varphi$ is identical with $\Gamma^k, \Pi \Rightarrow \Delta^k, \Sigma$. The induction step requires examination of all cases of possible derivations of $\varphi^k, \Pi \Rightarrow \Sigma$, and the role of the cut-formula in the transition. In cases where all occurrences of φ are parametric we simply apply the induction hypotheses to the premisses of $\varphi^k, \Pi \Rightarrow \Sigma$ and then apply the respective rule – it is essentially due to the context independence of almost all rules and the regularity of proofs, which together prevent violation of side conditions on eigenvariables. If one of the occurrences of φ in the premiss(es) is a side formula of the last rule we must additionally apply weakening to restore the missing formula before the application of the relevant rule.

In cases where one occurrence of φ in $\varphi^k, \Pi \Rightarrow \Sigma$ is principal we make use of the fact that φ in the left premiss is also principal; for the cases of contraction and weakening it is trivial. Note that due to condition that φ is principal in the left premiss it must be compound, since all rules introducing atomic formulae as principal are working only in the antecedents. Hence all cases where one occurrence of atomic φ in the right premiss would be introduced by means of $(R), (S), (T), (E)$ are not considered in the proof of this lemma. The only exceptions are axiomatic sequents $\Gamma \Rightarrow \Delta, \varphi$ with principal atomic φ, but they do not make any harm. □

Lemma 5 (Left reduction). *Let $\mathcal{D}_1 \vdash \Gamma \Rightarrow \Delta, \varphi^k$ and $\mathcal{D}_2 \vdash \varphi, \Pi \Rightarrow \Sigma$ with $d\mathcal{D}_1, d\mathcal{D}_2 < d\varphi$, then we can construct a proof $\mathcal{D}$ such that $\mathcal{D} \vdash \Gamma, \Pi^k \Rightarrow \Delta, \Sigma^k$ and $d\mathcal{D} < d\varphi$.*

Proof. By induction on the height of $\mathcal{D}_1$ but with some important differences. First note that we do not require φ to be principal in $\varphi, \Pi \Rightarrow \Sigma$ so it includes the case with φ atomic. In all these cases we just apply the induction hypothesis. This guarantees that even if an atomic cut formula was introduced in the right premiss by one of the rules $(R), (S), (T), (E)$ the reduction of the height is done only on the left premiss, and we always obtain the expected result. Now, in cases where one occurrence of φ in $\Gamma \Rightarrow \Delta, \varphi^k$ is principal we first apply the induction hypothesis to eliminate all other $k-1$ occurrences of φ in premisses and then we apply the respective rule. Since the only new occurrence of φ is principal we can make use of the right reduction lemma again and obtain the result, possibly after some applications of structural rules. □

Now we are ready to prove the cut elimination theorem:

Theorem 3. *Every proof in GO can be transformed into cut-free proof.*

Proof. By double induction: primary on $d\mathcal{D}$ and subsidiary on the number of maximal cuts (in the basis and in the inductive step of the primary induction). We always take the topmost maximal cut and apply Lemma 5 to it. By successive repetition of this procedure we diminish either the degree of a proof or the number of cuts in it until we obtain a cut-free proof. □

As a consequence of the cut elimination theorem for GO we obtain:

Corollary 1. *If* $\vdash \Gamma \Rightarrow \Delta$*, then it is provable in a proof which is closed under subformulae of* $\Gamma \cup \Delta$ *and atomic formulae.*

So cut-free GO satisfies the form of the subformula property which holds for several elementary theories as formalised by Negri and von Plato [14].

5 Modifications

Construction of rules which are deductively equivalent to axioms may be to some extent automatised (see e.g. Negri and von Plato [14], Braüner [1], or Marin, Miller, Pimentel and Volpe [12]). Still, even the choice of the version of (equivalent) axiom which will be used for transformation, may have an impact on the quality of obtained rules. Moreover, very often some additional tuning is necessary to obtain rules, which are well-behaved from the proof-theoretic point of view. In this section we will focus briefly on this problem and sketch some alternatives.

In our adequacy proofs we referred to the original formulation of LA, since rules $(R), (T), (S)$ correspond directly in a modular way to three conjuncts of $\mathrm{LA}^{\rightarrow}$. Our rule (E) however, is modelled not on $\mathrm{LA}^{\leftarrow}$ but rather on the suitable implication of variant 3 of LA from Lemma 1. As a first approximation we can obtain the rule:

$$\frac{\Gamma \Rightarrow \Delta, \exists z(\forall v(va \leftrightarrow vz) \wedge zb)}{\Gamma \Rightarrow \Delta, ab}$$

which after further decomposition and quantifier elimination yields:

$$\frac{da, \Gamma \Rightarrow \Delta, dc \quad dc, \Gamma \Rightarrow \Delta, da \quad \Gamma \Rightarrow \Delta, cb}{\Gamma \Rightarrow \Delta, ab}$$

(where d is a new parameter) which is very similar to (E) but with some active atoms in the succedents. This is troublesome for proving cut elimination if ab is a cut formula and a principal formula of $(R), (S)$ or (T) in the right premiss of cut. Fortunately, (E) is interderivable with this rule (it follows from the rule generation theorem in Indrzejczak [5]) and has the principal formula in the antecedent.

It is clear that if we focus on other variants then we can obtain different rules by their decomposition. In effect note that instead of (E) we may equivalently use the following rules based directly on LA, or on variants 2 and 1 respectively:

$$(E_{LA}) \quad \frac{da, \Gamma \Rightarrow \Delta, db \quad da, ea, \Gamma \Rightarrow \Delta, de \quad ab, \Gamma \Rightarrow \Delta}{ca, \Gamma \Rightarrow \Delta}$$

$$(E_2) \quad \frac{da, \Gamma \Rightarrow \Delta, dc \quad da, \Gamma \Rightarrow \Delta, cd \quad ab, \Gamma \Rightarrow \Delta}{ca, cb, \Gamma \Rightarrow \Delta}$$

$$(E_1) \quad \frac{da, ea, \Gamma \Rightarrow \Delta, de \quad ab, \Gamma \Rightarrow \Delta}{ca, cb, \Gamma, \Rightarrow \Delta}$$

where d, e are new parameters (eigenvariables).

Note, that each of these rules, used instead of (E), yields a variant of GO for which we can also prove cut elimination. However, as we will show by the end of this section, (E) seems to be optimal. Perhaps, the last one is the most economical in the sense of branching factor. However, since its left premiss directly corresponds to the condition $\forall xy(xa \wedge ya \rightarrow xy)$ it introduces two different new parameters to premisses which makes it more troublesome in some respects. In fact, if we want to reduce the branching factor it is possible to replace all these rules by the following variants:

$$(E') \quad \frac{da, \Gamma \Rightarrow \Delta, dc \quad dc, \Gamma \Rightarrow \Delta, da}{cb, \Gamma \Rightarrow \Delta, ab}$$

$$(E'_{LA}) \quad \frac{da, \Gamma \Rightarrow \Delta, db \quad da, ea, \Gamma \Rightarrow \Delta, de}{ca, \Gamma \Rightarrow \Delta, ab}$$

$$(E'_2) \quad \frac{da, \Gamma \Rightarrow \Delta, dc \quad da, \Gamma \Rightarrow \Delta, cd}{ca, cb, \Gamma \Rightarrow \Delta, ab}$$

$$(E'_1) \quad \frac{da, ea, \Gamma \Rightarrow \Delta, de}{ca, cb, \Gamma \Rightarrow \Delta, ab}$$

with the same proviso on eigenvariables d, e. Their interderivability with the rules stated first is easily obtained by means of the rule generation theorem too. These rules seem to be more convenient for proof search. However, for these primed rules cut elimination cannot be proved in the constructive way, for the reasons mentioned above, and it is an open problem if cut-free systems with these rules as primitive are complete.

We finish this section with stating the last reason for choosing (E). Let us explain why (E), the most complicated specific rule of GO, was claimed to be connected with extensionality. Consider the following two principles:

$WE\ \forall x(xa \leftrightarrow xb) \rightarrow \forall x(ax \leftrightarrow bx)$
$WExt\ \forall x(xa \leftrightarrow xb) \rightarrow \forall x(\varphi(x, a) \leftrightarrow \varphi(x, b))$

where $\varphi(x, a)$ denotes arbitrary formula with at least one occurrence of x (not bound by any quantifier within φ) and a.

Lemma 6. *WE is equivalent to $WExt$.*

Proof. That WE follows from $WExt$ is obvious since the former is a specific instance of the latter. The other direction is by induction on the complexity of φ. In the basis there are just two cases: $\varphi(x, a)$ is either xa or ax; the former is trivial and the latter is just WE. The induction step goes like an ordinary proof of the extensionality principle in FOL. □

Lemma 7. *In G (E) is equivalent to (WE).*

Proof. Note first that in G the following sequents are provable:

- $\forall x(ax \leftrightarrow cx), cb \Rightarrow ab$
- $\forall x(xa \leftrightarrow xc), da \Rightarrow dc$
- $\forall x(xa \leftrightarrow xc), dc \Rightarrow da$

we will use them in the proofs to follow.

For derivability of (E):

$$(Cut)\ \dfrac{(Cut)\ \dfrac{(\Rightarrow\forall)\ \dfrac{(\Rightarrow\leftrightarrow)\ \dfrac{da,\varGamma\Rightarrow\varDelta,dc \qquad dc,\varGamma\Rightarrow\varDelta,da}{\varGamma\Rightarrow\varDelta,da\leftrightarrow dc}}{\varGamma\Rightarrow\varDelta,\forall x(xa\leftrightarrow xc)} \qquad \mathcal{D}}{\varGamma\Rightarrow\varDelta,\forall x(ax\leftrightarrow cx)} \qquad \forall x(ax\leftrightarrow cx),cb\Rightarrow ab}{cb,\varGamma\Rightarrow\varDelta,ab}$$

where $\mathcal{D}$ is a proof of $\forall x(xa\leftrightarrow xc)\Rightarrow\forall x(ax\leftrightarrow cx)$ from WE and the rightmost sequent is provable. The endsequent by cut with $ab,\varGamma\Rightarrow\varDelta$ yields the conclusion of (E).

Provability of WE in G with (E):

$$(E)\ \dfrac{\forall x(xa\leftrightarrow xc),da\Rightarrow dc \qquad \forall x(xa\leftrightarrow xc),dc\Rightarrow da \qquad ab\Rightarrow ab}{\forall x(xa\leftrightarrow xc),cb\Rightarrow ab}$$

In the same way we prove $\forall x(xa\leftrightarrow xc),ab\Rightarrow cb$ which by $(\Rightarrow\leftrightarrow)$, $(\Rightarrow\forall)$ and $(\Rightarrow\rightarrow)$ yields WE.

□

This shows that we can obtain the axiomatization of elementary ontology by means of LA$^{\rightarrow}$ and WE (or $WExt$). Also instead of LA$^{\rightarrow}$ we can use three axioms corresponding to our three rules $(R),(S),(T)$. Note that if we get rid of (E) (or WE) we obtain a weaker version of ontology investigated by Takano [18]. If we get rid of quantifier rules we obtain a quantifier-free version of this system investigated by Ishimoto and Kobayashi [6].

On the basis of the specific features of sequent calculus we can obtain here for free also the intuitionistic version of ontology. As is well known it is sufficient to restrict the rules of G to sequents having at most one formula in the succedent (which requires small modifications like replacement of $(\leftrightarrow\Rightarrow)$ and $(\Rightarrow\vee)$ with two variants having always one side formula in the succedent) to obtain the version adequate for the intuitionistic FOL. Since all specific rules for ε can be restricted in a similar way, we can obtain the calculus GIO for the intuitionistic version of elementary ontology. One can easily check that all proofs showing the adequacy of GO and the cut elimination theorem are either intuitionistically correct or can be easily changed into such proofs. The latter remark concerns these proofs in which the classical version of $(\leftrightarrow\Rightarrow)$ required the introduction of the second side formula into succedent by $(\Rightarrow W)$; the intuitionistic two versions of $(\leftrightarrow\Rightarrow)$ do not require this step.

6 Extensions

Leśniewski and his followers were often working on ontology enriched with definitions of special predicates and name-creating functors. In this section we focus on a number of unary and binary predicates which are popular ontological constants. Instead of adding these definitions to GO we will introduce predicates by means of sequent rules satisfying conditions formulated for well-behaved SC rules. Let us call L_p the language of L_o enriched with all these predicates and GOP, the calculus with the additional rules for predicates. The definitions of the most important unary predicates are:

$$\begin{array}{ll} Da := \exists x(xa) & Va := \neg\exists x(xa) \\ Sa := \exists x(ax) & Ga := \exists xy(xa \wedge ya \wedge \neg xy) \end{array}$$

D, V, S, G are unary predicates informing that a is denoting, empty (or void), singular or general. D and S are Leśniewski's *ex* and *ob* respectively. He preferred also to apply $sol(a)$ which we symbolize with U (for unique):

$Ua := \forall xy(xa \wedge ya \rightarrow xy)$ [or simply $\neg Ga$]

The additional rules for these predicates are of the form:

$$(D \Rightarrow)\ \frac{ba, \Gamma \Rightarrow \Delta}{Da, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow D)\ \frac{\Gamma \Rightarrow \Delta, ca}{\Gamma \Rightarrow \Delta, Da} \qquad (S \Rightarrow)\ \frac{ab, \Gamma \Rightarrow \Delta}{Sa, \Gamma \Rightarrow \Delta}$$

$$(\Rightarrow S)\ \frac{\Gamma \Rightarrow \Delta, ac}{\Gamma \Rightarrow \Delta, Sa} \qquad (V \Rightarrow)\ \frac{\Gamma \Rightarrow \Delta, ca}{Va, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow V)\ \frac{ba, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, Va}$$

where b is new and c arbitrary in all schemata.

$$(G \Rightarrow)\ \frac{ba, ca, \Gamma \Rightarrow \Delta, bc}{Ga, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow G)\ \frac{\Gamma \Rightarrow \Delta, da \quad \Gamma \Rightarrow \Delta, ea \quad de, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, Ga}$$

$$(\Rightarrow U)\ \frac{ba, ca, \Gamma \Rightarrow \Delta, bc}{\Gamma \Rightarrow \Delta, Ua} \qquad (U \Rightarrow)\ \frac{\Gamma \Rightarrow \Delta, da \quad \Gamma \Rightarrow \Delta, ea \quad de, \Gamma \Rightarrow \Delta}{Ua, \Gamma \Rightarrow \Delta}$$

where b, c are new, and d, e are arbitrary parameters.

The binary predicates of identity, (weak and strong) coextensiveness, nonbeing b, subsumption and antysubsumption are defined in the following way:

$$\begin{array}{ll} a = b := ab \wedge ba & a\bar{\varepsilon}b := aa \wedge \neg ab \\ a \equiv b := \forall x(xa \leftrightarrow xb) & a \subset b := \forall x(xa \rightarrow xb) \\ a \approx b := a \equiv b \wedge Da & a \not\subset b := \forall x(xa \rightarrow \neg xb) \end{array}$$

Finally note that Aristotelian categorical sentences can be also defined in Leśniewski's ontology:

$$\begin{array}{ll} aAb := a \subset b \wedge Da & aEb := a \not\subset b \wedge Da \\ aIb := \exists x(xa \wedge xb) & aOb := \exists x(xa \wedge \neg xb) \end{array}$$

The rules for binary predicates:

$$(=\Rightarrow)\ \frac{ab, ba, \Gamma \Rightarrow \Delta}{a = b, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow=)\ \frac{\Gamma \Rightarrow \Delta, ab \quad \Gamma \Rightarrow \Delta, ba}{\Gamma \Rightarrow \Delta, a = b}$$

$$(\equiv\Rightarrow)\ \frac{\Gamma \Rightarrow \Delta, ca, cb \quad ca, cb, \Gamma \Rightarrow \Delta}{a \equiv b, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow\equiv)\ \frac{da, \Gamma \Rightarrow \Delta, db \quad db, \Gamma \Rightarrow \Delta, da}{\Gamma \Rightarrow \Delta, a \equiv b}$$

$$(\approx\Rightarrow)\ \frac{da, \Gamma \Rightarrow \Delta, ca, cb \quad ca, cb, da, \Gamma \Rightarrow \Delta}{a \approx b, \Gamma \Rightarrow \Delta}$$

$$(\Rightarrow\approx)\ \dfrac{da, \Gamma \Rightarrow \Delta, db \qquad db, \Gamma \Rightarrow \Delta, da \qquad \Gamma \Rightarrow \Delta, ca}{\Gamma \Rightarrow \Delta, a \approx b}$$

$$(\bar{\varepsilon} \Rightarrow)\ \dfrac{aa, \Gamma \Rightarrow \Delta, ab}{a\bar{\varepsilon}b, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow \bar{\varepsilon})\ \dfrac{\Gamma \Rightarrow \Delta, aa \qquad ab, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, a\bar{\varepsilon}b}$$

$$(\subset\Rightarrow)\ \dfrac{\Gamma \Rightarrow \Delta, ca \qquad cb, \Gamma \Rightarrow \Delta}{a \subset b, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow\subset)\ \dfrac{da, \Gamma \Rightarrow \Delta, db}{\Gamma \Rightarrow \Delta, a \subset b}$$

$$(\not\subset\Rightarrow)\ \dfrac{\Gamma \Rightarrow \Delta, ca \qquad \Gamma \Rightarrow \Delta, cb}{a \not\subset b, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow\not\subset)\ \dfrac{da, db, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, a \not\subset b}$$

$$(A \Rightarrow)\ \dfrac{da, \Gamma \Rightarrow \Delta, ca \qquad cb, da, \Gamma \Rightarrow \Delta}{aAb, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow A)\ \dfrac{da, \Gamma \Rightarrow \Delta, db \qquad \Gamma \Rightarrow \Delta, ca}{\Gamma \Rightarrow \Delta, aAb}$$

$$(E \Rightarrow)\ \dfrac{da, \Gamma \Rightarrow \Delta, ca \qquad da, \Gamma \Rightarrow \Delta, cb}{aEb, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow E)\ \dfrac{da, db, \Gamma \Rightarrow \Delta \qquad \Gamma \Rightarrow \Delta, ca}{\Gamma \Rightarrow \Delta, aEb}$$

$$(I \Rightarrow)\ \dfrac{da, db, \Gamma \Rightarrow \Delta}{aIb, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow I)\ \dfrac{\Gamma \Rightarrow \Delta, ca \qquad \Gamma \Rightarrow \Delta, cb}{\Gamma \Rightarrow \Delta, aIb}$$

$$(O \Rightarrow)\ \dfrac{da, \Gamma \Rightarrow \Delta, db}{aOb, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow O)\ \dfrac{\Gamma \Rightarrow \Delta, ca \qquad cb, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, aOb}$$

where d is new and c arbitrary (but c can be identical to d in rules for $\approx, A, E$).

Proofs of interderivability with equivalences corresponding to suitable definitions are trivial in most cases. We provide only one for the sake of illustration. The hardest case is $\approx$.

$$(\Rightarrow \forall)\ \dfrac{(\Rightarrow\leftrightarrow)\ \dfrac{(\approx\Rightarrow)\ \dfrac{da, ca \Rightarrow ca, cb \qquad da, ca, cb \Rightarrow cb}{a \approx b, ca \Rightarrow cb} \qquad \dfrac{da, ca \Rightarrow ca, cb \qquad da, ca, cb \Rightarrow ca}{a \approx b, cb \Rightarrow ca}}{a \approx b \Rightarrow ca \leftrightarrow cb}}{a \approx b \Rightarrow \forall x(xa \leftrightarrow xb)}$$

and

$$(\approx\Rightarrow)\ \dfrac{(\Rightarrow \exists)\ \dfrac{ca \Rightarrow ca, aa, ab}{ca \Rightarrow \exists x(xa), aa, ab} \qquad (\Rightarrow \forall)\ \dfrac{ca, aa, ab \Rightarrow ca}{ca, aa, ab \Rightarrow \exists x(xa)}}{a \approx b \Rightarrow \exists x(xa)}$$

by $(\Rightarrow \wedge)$ yield one part. For the second:

$$(\wedge \Rightarrow)\ \dfrac{(\exists \Rightarrow)\ \dfrac{(\Rightarrow\approx)\ \dfrac{\forall x(xa \leftrightarrow xb), da \Rightarrow db \qquad \forall x(xa \leftrightarrow xb), db \Rightarrow da \qquad ca \Rightarrow ca}{\forall x(xa \leftrightarrow xb), ca \Rightarrow a \approx b}}{\forall x(xa \leftrightarrow xb), \exists x(xa) \Rightarrow a \approx b}}{\forall x(xa \leftrightarrow xb) \wedge \exists x(xa) \Rightarrow a \approx b}$$

where the left and the middle premiss are obviously provable by means of $(\forall \Rightarrow)$, $(\leftrightarrow\Rightarrow)$. We omit proofs of the derivability of both rules in GO enriched with the axiom $\Rightarrow \forall x(xa \leftrightarrow xb) \wedge \exists x(xa) \leftrightarrow a \approx b$.

We treat all these predicates as new constants hence their complexity is fixed as 1, in contrast to atomic formulae, which are of complexity 0. Of course we can consider ontology with an arbitrary selection of these predicates according to the needs. Accordingly we can enrich GO also with arbitrary selection of suitable rules for predicates. All the results holding for GOP are correct for any subsystem. Let us list some important features of these rules and enriched GO:

1. All rules for predicates are explicit, separate and symmetric, which are usual requirements for well-behaved rules in sequent calculi (see e.g. [5]). In this respect they are similar to the rules for logical constants and differ from specific rules for ε which are one-sided (in the sense of having principal formulae always in the antecedent).
2. All these new rules satisfy the subformula property in the sense that side formulae are only atomic.
3. The substitution lemma holds for GO with any combination of the above rules.
4. All rules are pairwise reductive, modulo substitution of terms,

We do not prove the substitution lemma, since the proof is standard, but we comment on the last point, since cut elimination holds due to 3 and 4. The notion of reductivity for sequent rules was introduced by Ciabattoni [2] and it may be roughly defined as follows: A pair of introduction rules ($\Rightarrow \star$), ($\star \Rightarrow$) for a constant $\star$ is reductive if an application of cut on cut formulae introduced by these rules may be replaced by the series of cuts made on less complex formulae, in particular on their subformulae. Basically it enables the reduction of cut-degree in the proof of cut elimination. Again we illustrate the point with respect to the most complicated case. Let us consider the application of cut with the cut formula $a \approx b$, then the left premiss of this cut was obtained by:

$$(\Rightarrow\approx)\ \frac{ca, \Gamma \Rightarrow \Delta, cb \qquad cb, \Gamma \Rightarrow \Delta, ca \qquad \Gamma \Rightarrow \Delta, da}{\Gamma \Rightarrow \Delta, a \approx b}$$

where c is new and d is arbitrary. And the right premiss was obtained by:

$$(\approx\Rightarrow)\ \frac{ea, \Pi \Rightarrow \Sigma, fa, fb \qquad ea, fa, fb, \Pi \Rightarrow \Sigma}{a \approx b, \Pi \Rightarrow \Sigma}$$

where e is new and f is arbitrary.

By the substitution lemma on the premisses of $(\Rightarrow\approx), (\approx\Rightarrow)$ we obtain:

1. $fa, \Gamma \Rightarrow \Delta, fb$
2. $fb, \Gamma \Rightarrow \Delta, fa$
3. $da, \Pi \Rightarrow \Sigma, fa, fb$
4. $da, fa, fb, \Pi \Rightarrow \Sigma$

and we can derive:

$$(C)\ \dfrac{(Cut)\ \dfrac{(C)\ \dfrac{(Cut)\ \dfrac{(Cut)\ \dfrac{\Gamma \Rightarrow \Delta, da \qquad da, \Pi \Rightarrow \Sigma, fa, fb}{\Gamma, \Pi \Rightarrow \Delta, \Sigma, fa, fb} \qquad fb, \Gamma \Rightarrow \Delta, fa}{\Gamma, \Gamma, \Pi \Rightarrow \Delta, \Delta, fa, fa}}{\Gamma, \Pi \Rightarrow \Delta, \Sigma, fa} \qquad \mathcal{D}}{\Gamma, \Gamma, \Pi, \Pi \Rightarrow \Delta, \Delta, \Sigma, \Sigma}}{\Gamma, \Pi \Rightarrow \Delta, \Sigma}$$

where $\mathcal{D}$ is a similar proof of $fa, \Gamma, \Pi \Rightarrow \Delta, \Sigma$ from $\Gamma \Rightarrow \Delta, da$, 4 and 1 by cuts and contractions. All cuts are of lower degree than the original cut. It is routine exercise to check that all rules for predicates are reductive and this is sufficient for proving Lemma 4 and 5 for GOP. As a consequence we obtain:

Theorem 4. *Every proof in GOP can be transformed into cut-free proof.*

Since the rules are modular this holds for every subsystem based on a selection of the above rules.

7 Conclusion

Both the basic system GO and its extension GOP are cut-free and satisfy a form of the subformula property. It shows that Leśniewski's ontology admits standard proof-theoretical study and allows us to obtain reasonable results. In particular, we can prove for GO the interpolation theorem using the Maehara strategy (see e.g. [19]) and this implies for GO other expected results like e.g. Beth's definability theorem. Space restrictions forbid to present it here. On the other hand, we restricted our study to the system with simple names only, whereas fuller study should cover also complex names built with the help of several name-forming functors. The typical ones are the counterparts of the well-known class operations definable in Leśniewski's ontology in the following way:

$$a\bar{b} := aa \wedge \neg ab \quad a(b \cap c) := ab \wedge ac \quad a(b \cup c) := ab \vee ac$$

It is not a problem to provide suitable rules corresponding to these definitions:

$$(- \Rightarrow)\ \frac{aa, \Gamma \Rightarrow \Delta, ab}{a\bar{b}, \Gamma \Rightarrow \Delta} \qquad (\Rightarrow -)\ \frac{ab, \Gamma \Rightarrow \Delta \quad \Gamma \Rightarrow \Delta, aa}{\Gamma \Rightarrow \Delta, a\bar{b}}$$

$$(\cap \Rightarrow)\ \frac{ab, ac, \Gamma \Rightarrow \Delta}{a(b \cap c), \Gamma \Rightarrow \Delta} \qquad (\Rightarrow \cap)\ \frac{\Gamma \Rightarrow \Delta, ab \quad \Gamma \Rightarrow \Delta, ac}{\Gamma \Rightarrow \Delta, a(b \cap c)}$$

$$(\cup \Rightarrow)\ \frac{ab, \Gamma \Rightarrow \Delta \quad ac, \Gamma \Rightarrow \Delta}{a(b \cup c), \Gamma \Rightarrow \Delta} \qquad (\Rightarrow \cup)\ \frac{\Gamma \Rightarrow \Delta, ab, ac}{\Gamma \Rightarrow \Delta, a(b \cup c)}$$

Although their structure is similar to the rules provided for predicates in the last section, their addition raises important problems. One is of a more general nature and well-known: definitions of term-forming operations in ontology are creative. Although it was intended in the original architecture of Leśniewski's systems, in the modern approach this is not welcome. Iwanuś [7] has shown that the problem can be overcome by enriching elementary ontology with two axioms corresponding to special versions of the comprehension axiom but this opens a problem of derivability of these axioms in GO enriched with special rules.

There is also a specific problem with cut elimination for GO with added complex terms and suitable rules. Even if they are reductive (and the rules stated above are reductive, as a reader can check), we run into a problem with quantifier rules. If unrestricted instantiation of terms is admitted in $(\Rightarrow \exists)$, $(\forall \Rightarrow)$ the subformula property is lost. One can find some solutions for this problem, for example by using two separated measures of complexity for formula-makers and term-makers (see e.g. [3]), or by restricting in some way the instantiation of terms in respective quantifier rules (see e.g. [4]). The examination of these possibilities is left for further study.

The last open problem deserving careful study is the possibility of application for automated proof-search and obtaining semi-decision procedures (or decision procedures for quantifier-free subsystems) on the basis of the provided sequent calculus. In particular, due to modularity of provided rules, one could obtain in this way decision procedures for several quantifier-free subsystems investigated by Pietruszczak [15], or by Ishimoto and Kobayashi [6].

Acknowledgements. The research is supported by the National Science Centre, Poland (grant number: DEC-2017/25/B/HS1/01268). I am also greatly indebted to Nils Kürbis for his valuable comments.

References

1. Braüner, T.: Hybrid Logic and its Proof-Theory. Springer, Cham (2011). https://doi.org/10.1007/978-94-007-0002-4
2. Ciabattoni, A.: Automated generation of analytic calculi for logics with linearity. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 503–517. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30124-0_38
3. Indrzejczak, A.: Fregean description theory in proof-theoretic setting. Logic Log. Philos. **28**(1), 137–155 (2019)
4. Indrzejczak, A.: Free logics are cut-free. Stud. Log. **109**(4), 859–886 (2021)
5. Indrzejczak, A.: Sequents and Trees. An Introduction to the Theory and Applications of Propositional Sequent Calculi. Birkhäuser (2021)
6. Ishimoto, A., Kobayashi, M.: A propositional fragment of Leśniewski's ontology and its formulation by the tableau method. Stud. Log. **41**(2–3), 181–196 (1982)
7. Iwanuś, B.: On Leśniewski's elementary ontology. Stud. Log. **31**(1), 73–119 (1973)
8. Küng, G., Canty, J.T.: Substitutional quantification and Leśniewskian quantifiers. Theoria **36**, 165–182 (1970)
9. Leśniewski, S.: Über Funktionen, deren Felder Gruppen mit Rücksicht auf diese Funktionen sind. Fundam. Math. **13**, 319–332 (1929). [English translation. In: Leśniewski [11]]
10. Leśniewski, S.: Über Funktionen, deren Felder Abelsche Gruppen in Bezug auf diese Funktionen sind. Fundam. Math. **14**, 242–251 (1929). [English translation. In: Leśniewski [11]]
11. Leśniewski, S.: Collected Works, vol. II. Surma, S., Srzednicki, J., Barnett, D.I. Kluwer/PWN (1992)
12. Marin, S., Miller, D., Pimentel, E., Volpe, M.: From axioms to synthetic inference rules via focusing. Ann. Pure Appl. Log. **173**(5), 103091 (2022)
13. Metcalfe, G., Olivetti, N., Gabbay, D.: Proof Theory for Fuzzy Logics. Springer, Heidelberg (2008). https://doi.org/10.1007/978-1-4020-9409-5
14. Negri, S., von Plato, J.: Structural Proof Theory. Cambridge University Press, Cambridge (2001)
15. Pietruszczak, A.: Quantifier-free Calculus of Names. Systems and Metatheory, Toruń (1991). [in Polish]
16. Rickey, F.: Interpretations of Leśniewski's ontology. Dialectica **39**(3), 181–192 (1985)

17. Słupecki, J.: S. Leśniewski's calculus of names. Stud. Log. **3**(1), 7–72 (1955)
18. Takano, M.: A semantical investigation into Leśniewski's axiom and his ontology. Stud. Log. **44**(1), 71–78 (1985)
19. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Oxford University Press, Oxford (1996)
20. Urbaniak, R.: Leśniewski's Systems of Logic and Foundations of Mathematics. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-00482-2

Bayesian Ranking for Strategy Scheduling in Automated Theorem Provers

Chaitanya Mangla(✉), Sean B. Holden, and Lawrence C. Paulson

Computer Laboratory, University of Cambridge, Cambridge, England
{cm772,sbh11,lp15}@cl.cam.ac.uk

Abstract. A *strategy schedule* allocates time to proof strategies that are used in sequence in a theorem prover. We employ Bayesian statistics to propose alternative sequences for the strategy schedule in each proof attempt. Tested on the TPTP problem library, our method yields a time saving of more than 50%. By extending this method to optimize the fixed time allocations to each strategy, we obtain a notable increase in the number of theorems proved.

Keywords: Bayesian machine learning · Strategy scheduling · Automated theorem proving

1 Introduction

Theorem provers have wide-ranging applications, including formal verification of large mathematical proofs [9] and reasoning in knowledge-bases [37]. Thus, improvements in provers that lead to more successful proofs, and savings in the time taken to discover proofs, are desirable.

Automated theorem provers generate proofs by utilizing inference procedures in combination with heuristic search. A specific configuration of a prover, which may be specialized for a certain class of problems, is termed a *strategy*. Provers such as E [27] can select from a portfolio of strategies to solve the goal theorem. Furthermore, certain provers hedge their allocated proof time across a number of proof strategies by use of a *strategy schedule*, which specifies a time allocation for each strategy and the sequence in which they are used until one proves the goal theorem. This method was pioneered in the Gandalf prover [33].

Prediction of the effectiveness of a strategy prior to a proof attempt is usually intractable or undecidable [12]. A practical implementation must infer such a prediction by tractable approximations. Therefore, machine learning methods for strategy invention, selection and scheduling are actively researched. Machine learning methods for strategy selection conditioned on the proof goal have shown promising results [3]. Good results have also been reported for strategy synthesis using machine learning [1]. Work on machine learning for algorithm portfolios—which allocate resources to multiple solvers simultaneously—is also relevant to strategy scheduling because of its similar goals. For this purpose, Silverthorn and Miikkulainen propose latent class models [31] .

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 559–577, 2022.
https://doi.org/10.1007/978-3-031-10769-6_33

In this work, we present a method for generating strategy schedules using Bayesian learning with two primary goals: to reduce proving time or to prove more theorems. We have evaluated this method for both purposes using iLeanCoP, an intuitionistic first-order logic prover with a compact implementation and good performance [18]. Intuitionistic logic is a non-standard form of first-order logic, of which relatively little is known with regard to automation. It is of interest in theoretical computer science and philosophy of mathematics [7]. Among intuitionistic provers, iLeanCoP is seen as impressive and is able to prove a sufficient number of theorems in our benchmarks for significance testing. Its core is implemented in around thirty lines of Prolog; such simplicity adds clarity to interpretations of our results. Our method was benchmarked on the Thousands of Problems for Theorem Provers (TPTP) problem library [32], in which we are able to save more than 50% on proof time when aiming for the former goal. Towards the latter goal, we are able to prove notably more theorems.

Our two primary, complementary, contributions presented here are: first, a Bayesian machine learning model for strategy scheduling; and second, engineered features for use in that model. The text below is organized as follows. In Sect. 2, we introduce preliminary material used subsequently to construct a machine learning model for strategy scheduling, described in Sects. 3–7. The data used to train and evaluate this model are described in Sect. 8, followed by experiments, results and conclusions in Sects. 9–12.

2 Distribution of Permutations

We model a strategy schedule using a vector of strategies, and thus all schedules are *permutations* of the same.

Definition 1 (Permutation). *Let $M \in \mathbb{N}$. A permutation $\boldsymbol{\pi} \in \mathbb{N}^M$ is a vector of indices, with $\pi_i \in \{1, \ldots, M\}$ and $\forall i \neq j : \pi_i \neq \pi_j$, representing a reordering of the components of an M-dimensional vector $\boldsymbol{s}$ to $[s_{\pi_1}, s_{\pi_2}, \ldots, s_{\pi_M}]^\intercal$.*

In this text, vector-valued variables, such as $\boldsymbol{\pi}$ above, are in boldface, which must change when they are indexed, like π_1 for example. For probabilistic modelling of schedules represented using permutations, we use the Plakett-Luce model [14,21] to define a parametric probability distribution over permutations.

Definition 2 (Plakett-Luce distribution). *The Plakett-Luce distribution* $\mathrm{Perm}(\boldsymbol{\lambda})$ *with parameter $\boldsymbol{\lambda} \in \mathbb{R}^M_{>0}$, has support over permutations of indices* $\{1, \ldots, M\}$*. For permutation $\boldsymbol{\Pi}$ distributed as* $\mathrm{Perm}(\boldsymbol{\lambda})$,

$$\Pr(\boldsymbol{\Pi} = \boldsymbol{\pi}; \boldsymbol{\lambda}) = \prod_{j=1}^{M} \frac{\lambda_{\pi_j}}{\sum_{u=j}^{M} \lambda_{\pi_u}}.$$

In latter sections, we use the parameter $\boldsymbol{\lambda}$ to assign an abstract 'score' to strategies when modelling distributions over schedules. This score is particularly useful due to the following theorem.

Theorem 1. *Let* $\boldsymbol{\pi}^*$ *be a mode of the distribution* $\operatorname{Perm}(\boldsymbol{\lambda})$*, that is*

$$\boldsymbol{\pi}^* = \operatorname*{argmax}_{\boldsymbol{\pi}} \Pr(\boldsymbol{\pi}; \boldsymbol{\lambda}).$$

Then, $\lambda_{\pi_1^*} \geqslant \lambda_{\pi_2^*} \geqslant \lambda_{\pi_3^*} \geqslant \ldots \geqslant \lambda_{\pi_M^*}$.

Thus, assuming $\boldsymbol{\lambda}$ is a vector of the score of each strategy, the highest probability permutation indexes the strategies in decreasing order of scores. Conversely, the highest probability permutation can be obtained efficiently by sorting the indices of $\boldsymbol{\lambda}$ with respect to their corresponding values in decreasing order. Cao et al. [4] have presented a proof of Theorem 1, and Cheng et al. [5] have discussed some further interesting details.

Example 1. Let $\boldsymbol{\lambda} = [1, 9]^\intercal$, $\boldsymbol{\pi}^{(1)} = [1, 2]^\intercal$ and $\boldsymbol{\pi}^{(2)} = [2, 1]^\intercal$. Then,

$$\Pr(\boldsymbol{\Pi} = \boldsymbol{\pi}^{(1)}; \boldsymbol{\lambda}) = \frac{\lambda_{\boldsymbol{\pi}_1^{(1)}}}{\lambda_{\boldsymbol{\pi}_1^{(1)}} + \lambda_{\boldsymbol{\pi}_2^{(1)}}} \cdot \frac{\lambda_{\boldsymbol{\pi}_2^{(1)}}}{\lambda_{\boldsymbol{\pi}_2^{(1)}}} = \frac{1}{1+9} \cdot \frac{9}{9} = \frac{1}{10}.$$

Similarly, $\Pr(\boldsymbol{\Pi} = \boldsymbol{\pi}^{(2)}; \boldsymbol{\lambda}) = 9/10$. □

Theorem 2. $\operatorname{Perm}(c\boldsymbol{\lambda}) = \operatorname{Perm}(\boldsymbol{\lambda})$*, for any scalar constant* $c > 0$*.*

In other words, the Plakett-Luce distribution is invariant to the scale of the parameter vector.

Lemma 1. $\operatorname{Perm}(\exp(\boldsymbol{\lambda} + c)) = \operatorname{Perm}(\exp(\boldsymbol{\lambda}))$*, for any scalar constant* $c \in \mathbb{R}$*.*

Lemma 1 follows from Theorem 2, and shows the same distribution is translation invariant if the parameter is exponentiated. Cao et al. [4] give proofs of both.

3 A Maximum Likelihood Model

We model a strategy schedule as a ranking of known strategies, where each strategy is constructed by a parameter setting and time allocation. A ranking therein is a permutation of strategies, with each strategy retaining its time allocation irrespective of the ordering. We construct, in this section, a model for inference of such permutations that is linear in the parameters.

Suppose we have a repository of N theorems which we test against each of our M known strategies to build a data-set $\mathcal{D} = \{(\boldsymbol{\pi}^{(i)}, \boldsymbol{x}^{(i)})\}_{i=1}^{N}$, where $\boldsymbol{\pi}^{(i)}$ is a desirable ordering of strategies for theorem i and $\boldsymbol{x}^{(i)}$ is a feature vector representation of the theorem. In Sect. 9, we detail how we instantiated $\mathcal{D}$ for our experiments, which serves as an example for any other implementation. We assume that $\boldsymbol{\pi}^{(i)}$ has Plakett-Luce distribution conditioned on $\boldsymbol{x}^{(i)}$ such that

$$\Pr(\boldsymbol{\pi}; \boldsymbol{x}, \boldsymbol{\omega}) = \operatorname{Perm}(\boldsymbol{\Lambda}(\boldsymbol{x}, \boldsymbol{\omega})), \qquad (1)$$

where $\boldsymbol{\omega}$ is a parameter the model must learn and $\boldsymbol{\Lambda}(\cdot)$ is a vector-valued function of range $\mathbb{R}^M_{>0}$. We use the notation $\Lambda(\cdot)_i$ to index into the value of $\boldsymbol{\Lambda}(\cdot)$. We represent our prover strategies with feature vectors $\{\boldsymbol{d}^{(j)}\}_{j=1}^{M}$. To calculate the score of strategy j using $\Lambda(\cdot)_j$, we specify

$$\Lambda(\boldsymbol{x}^{(i)}, \boldsymbol{\omega})_j = \exp\left(\boldsymbol{\phi}(\boldsymbol{x}^{(i)}, \boldsymbol{d}^{(j)})^{\mathsf{T}} \boldsymbol{\omega}\right) \tag{2}$$

to ensure that the scores are positive valued, where $\boldsymbol{\phi}$ is a suitable basis expansion function. Assuming the data is i.i.d, the likelihood of the parameter vector is given by

$$\mathcal{L}(\boldsymbol{\omega}) = p(\mathcal{D}; \boldsymbol{\omega}) = \prod_{i=1}^{N} \Pr(\boldsymbol{\pi}^{(i)}; \boldsymbol{\Lambda}(\boldsymbol{x}^{(i)}, \boldsymbol{\omega})). \tag{3}$$

An $\hat{\boldsymbol{\omega}}$ that maximizes this likelihood can then be used to forecast the distribution over permutations for a new theorem $\boldsymbol{x}^*$ by evaluating $\mathrm{Perm}(\boldsymbol{\Lambda}(\boldsymbol{x}^*, \hat{\boldsymbol{\omega}}))$ for all permutations. This would incur factorial complexity; however, we are often only interested in the most likely permutation, which can be retrieved in polynomial time. Specifically for strategy scheduling the permutation with the highest predicted probability should reflect the orderings in the data. For this purpose, we use Theorem 1 to find the highest probability permutation $\boldsymbol{\pi}^*$ by sorting the values of $\{\boldsymbol{\Lambda}(\boldsymbol{x}^*, \hat{\boldsymbol{\omega}})_j\}_{j=1}^{M}$ in descending order.

Remark 1. A method named ListNet designed to rank documents for search queries using the Plakett-Luce distribution is evaluated by Cao et al. [4]. Their evaluation uses a linear basis expansion. We can derive a similar construction in our model by setting

$$\boldsymbol{\phi}(\boldsymbol{x}^{(i)}, \boldsymbol{d}^{(j)}) = [\boldsymbol{x}^{(i)\mathsf{T}}, \boldsymbol{d}^{(j)\mathsf{T}}]^{\mathsf{T}}. \tag{4}$$

Remark 2. The likelihood in Equation (3) can be maximized by minimizing the negative log likelihood $\ell(\boldsymbol{\omega}) = -\log \mathcal{L}(\boldsymbol{\omega})$, which (as shown by Schäfer and Hüllermeier [26]) is convex and therefore can be minimized using gradient-based methods. The minima may, however, be unidentifiable due to translation invariance, as demonstrated by Lemma 1. This problem is eliminated in our Bayesian model by the use of a Gaussian prior, as explained in Sect. 4.

Example 2. Let there be $N = 2$ theorems and $M = 2$ strategies. Let the theorems and strategies be characterized by univariate values such that $x^{(1)} = 1$, $x^{(2)} = 2$, $d^{(1)} = 1$ and $d^{(2)} = 2$.

Suppose strategy $d^{(1)}$ is ideal for theorem $x^{(1)}$ and strategy $d^{(2)}$ for $x^{(2)}$, as shown on the right, where a + indicates the preferred strategy.

	$d^{(1)}$	$d^{(2)}$
$x^{(1)}$	+	−
$x^{(2)}$	−	+

This is evidently an example of a parity problem [34], and hence cannot be modelled by a simple linear expansion using the basis function mentioned in Remark 1. A solution in this instance is to use

$$\phi(x^{(i)}, d^{(j)}) = x^{(i)} \cdot d^{(j)}.$$

The parameter ω is then one-dimensional, and the required training data takes the form $\mathcal{D} = \{([1, 2]^{\mathsf{T}}, 1), ([2, 1]^{\mathsf{T}}, 2)\}$. We find that $\mathcal{L}(w)$ is convex, with maxima at $\hat{\omega} = 0.42$ as shown in Fig. 1.

□

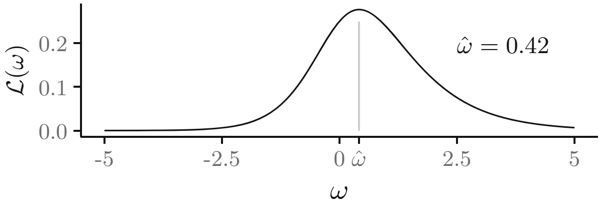


Fig. 1. The likelihood function in Example 2.

4 Bayesian Inference

We place a Gaussian prior distribution on the parameter $\boldsymbol{\omega}$ of the model described in Sect. 3. This has two advantages: first, the posterior mode is identifiable, as noted by Johnson et al. [11] and demonstrated in Example 3 on page 7; second, the parameter is regularized. With this prior specified as the normal distribution

$$\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{m_0}, \boldsymbol{S_0}), \tag{5}$$

and assuming $\boldsymbol{\pi}$ is independent of $\mathcal{D}$ given $(\boldsymbol{x}, \boldsymbol{\omega})$, the posterior predictive distribution is

$$p(\boldsymbol{\pi}|\boldsymbol{x}^*, \mathcal{D}) = \int p(\boldsymbol{\pi}|\boldsymbol{x}^*, \boldsymbol{\omega}) p(\boldsymbol{\omega}|\mathcal{D}) \mathrm{d}\boldsymbol{\omega},$$

which may be approximated by sampling from the posterior,

$$\boldsymbol{\omega}^s \sim p(\boldsymbol{\omega}|\mathcal{D}), \tag{6}$$

to obtain

$$p(\boldsymbol{\pi}|\boldsymbol{x}^*, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^{S} p(\boldsymbol{\pi}|\boldsymbol{x}^*, \boldsymbol{\omega}^s). \tag{7}$$

Given a new theorem $\boldsymbol{x}^*$, to find the permutation of strategies with the highest probability of success, using the approximation above would require its evaluation for every permutation of $\boldsymbol{\pi}$. This process incurs factorial complexity. We instead make a Bayes point approximation [16] using the mean values of the samples such that,

$$\begin{aligned} p(\boldsymbol{\pi}|\boldsymbol{x}^*, \mathcal{D}) &\approx p(\boldsymbol{\pi}|\boldsymbol{x}^*, \langle \boldsymbol{\omega}^s \rangle) && \text{using Eq. (7)} \\ &= \Pr(\boldsymbol{\pi}|\boldsymbol{\Lambda}(\boldsymbol{x}^*, \langle \boldsymbol{\omega}^s \rangle)) && \text{using Eq. (1)}, \end{aligned}$$

where $\langle \cdot \rangle$ denotes mean value. The mean of the Plakett-Luce parameter for Bayesian inference has been used in prior work [8] to obtain good results. Furthermore, using that, the highest probability permutation can be obtained by using Theorem 1, thereby incurring only the cost of sorting the items. This saving is substantial when generating a strategy schedule, because it saves on prediction time, which is important for the following reason.

Algorithm 1. Metropolis-Hastings Algorithm

Suppose we have generated samples $\{\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(i)}\}$ from the *target distribution* p. Generate $\boldsymbol{\omega}^{(i+1)}$ as follows.

```
1: Generate candidate value $\dot{\boldsymbol{\omega}} \sim q(\boldsymbol{\omega}^{(i)})$, where $q$ is the proposal distribution.
2: Evaluate $r \equiv r(\boldsymbol{\omega}^{(i)}, \dot{\boldsymbol{\omega}})$ where
       $r(x, y) = \min\left\{ \frac{p(y)}{p(x)} \frac{q(x|y)}{q(y|x)}, 1 \right\}.$
3: Set
       $\boldsymbol{\omega}^{(i+1)} = \begin{cases} \dot{\boldsymbol{\omega}} & \text{with probability } r \\ \boldsymbol{\omega}^{(i)} & \text{with probability } 1 - r. \end{cases}$
```

Remark 3. While benchmarking and in typical use, a prover is allocated a fixed amount of time for a proof attempt, and any time taken to predict a strategy schedule must be accounted for within this allocation. Time taken for this prediction is time taken away from the prover itself which could have been invested in the proof search. Therefore, it is essential to minimize schedule prediction time. It is particularly wise to favour a saving in prediction time at the cost of model optimization and training time.

Remark 4. In our implementation we set $\boldsymbol{m}_0 = 0$. This has the effect of prioritizing smaller weights $\boldsymbol{\omega}$ in the posterior. Furthermore, we set $\boldsymbol{S}_0 = \eta \boldsymbol{I}, \eta \in \mathbb{R}$, where $\boldsymbol{I}$ is the identity matrix. Consequently, the hyperparameter η controls the strength of the prior, since the entropy of the Gaussian prior scales linearly by $\log |\boldsymbol{S}_0|$.

Remark 5. A specialization of the Plakett-Luce distribution using the Thurstonian interpretation admits a Gamma distribution conjugate prior [8]. That, however, is unavailable to our model when parametrized as shown in Eq. (1).

5 Sampling

We use the Markov chain Monte Carlo (MCMC) Metropolis-Hastings algorithm [38] to generate samples from the posterior distribution. In MCMC sampling, one constructs a Markov chain whose stationary distribution matches the target distribution p. For the Metropolis-Hastings algorithm, stated in Algorithm 1, this chain is constructed using a proposal distribution $y|x \sim q$, where q is set to a distribution that can be conveniently sampled from.

Note that while calculating r in Algorithm 1, the normalization constant of the target density p cancels out. This is to our advantage; to generate samples $\boldsymbol{\omega}^s$ from the posterior, which is, by Eq. (3) and Eq. (5),

$$\begin{aligned} p(\boldsymbol{\omega}|\mathcal{D}) &\propto p(\mathcal{D}|\boldsymbol{\omega}) p(\boldsymbol{\omega}) \\ &= \mathcal{L}(\boldsymbol{\omega}) \mathcal{N}(\boldsymbol{m_0}, \boldsymbol{S_0}), \end{aligned} \tag{8}$$

the posterior only needs to be computed in this unnormalized form.

In this work, we choose a random walk proposal of the form

$$q(\boldsymbol{\omega}'|\boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\omega}'|\boldsymbol{\omega}, \Sigma_q), \tag{9}$$

and tune Σ_q for efficient sampling simulation. We start the simulation at a local mode $\hat{\boldsymbol{\omega}}$, and set $\mathcal{N}(\hat{\boldsymbol{\omega}}, \Sigma_q)$ to approximate the local curvature of the posterior at that point using methods by Rossi [25]. Specifically, our procedure for computing Σ_q is as follows.

1. First, writing the posterior from Eq. (8) as

   $$p(\boldsymbol{\omega}|\mathcal{D}) = \frac{1}{Z} e^{-E(\boldsymbol{\omega})},$$

   where Z is the normalization constant, we have

   $$E(\boldsymbol{\omega}) = -\log \mathcal{L}(\boldsymbol{\omega}) - \log \mathcal{N}(\boldsymbol{m_0}, \boldsymbol{S_0}). \tag{10}$$

   We find a local mode $\hat{\boldsymbol{\omega}}$ by optimizing $E(\boldsymbol{\omega})$ using a gradient-based method.
2. Then, using a Laplace approximation [2], we approximate the posterior in the locality of this mode to

   $$\mathcal{N}(\hat{\boldsymbol{\omega}}, H^{-1}), \text{ where } H = \nabla\nabla E(\boldsymbol{\omega})|_{\hat{\boldsymbol{\omega}}}$$

   is the Hessian matrix of $E(\boldsymbol{\omega})$ evaluated at that local mode.
3. Finally, we set

   $$\Sigma_q = s^2 H^{-1}$$

   in Eq. (9), where s is used to tune all the length scales. We set this value to $s^2 = 2.38$ based on the results by Roberts and Rosenthal [24].

Remark 6. When calculating r in Algorithm 1 during sampling, to evaluate the unnormalized posterior at any point $\boldsymbol{\omega}^s$ we compute it from Equation (10) as $\exp(-E(\boldsymbol{\omega}^s))$—it is therefore the only form in which the posterior needs to be coded in the implementation.

Example 3 (Gaussian Prior). To demonstrate the effect of using a Gaussian prior, we build upon Example 2, with the data taking the form

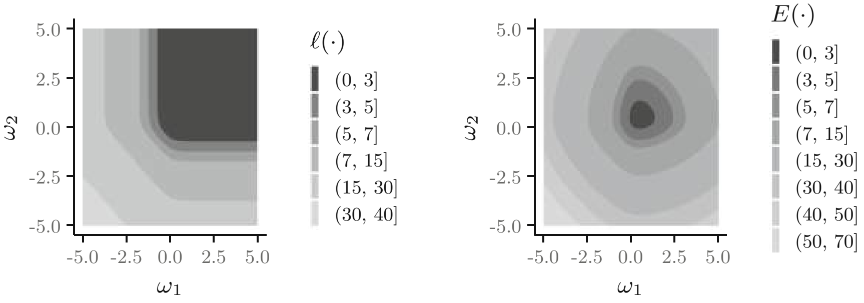
$$\mathcal{D} = \{([1, 2]^\intercal, 1), ([2, 1]^\intercal, 2)\}.$$

We perform basis expansion as explained in Sect. 6 with prior parameter $\eta = 1.0$, kernel $\sigma = 0.1$ and $\varsigma = 2$ centres. Thus, the model parameter is

$$\boldsymbol{\omega} = [\omega_1, \omega_2]^\intercal, \quad \boldsymbol{\omega} \in \mathbb{R}^2.$$

The unnormalized negative log posterior $E(\omega_1, \omega_2)$, as defined in Eq. (10), is shown in Fig. 2b; and the negative log likelihood $\ell(\omega_1, \omega_2) = -\log \mathcal{L}(\omega_1, \omega_2)$ as mentioned in Remark 2, is shown in Fig. 2a. Note the contrast in the shape of the two surfaces. The minimum is along the top-right portion in Fig. 2a, which is flat and leads to an unidentifiable point estimate, whereas in Fig. 2b, the minimum is in a narrow region near the centre. The Gaussian prior, in informal terms, has lifted the surface up, with an effect that increases in proportion to the distance from the origin.

□



(a) Likelihood function $\ell(\omega_1, \omega_2)$

(b) Posterior function $E(\omega_1, \omega_2)$

Fig. 2. Comparison of the shape of the likelihood and the posterior functions.

6 Basis Expansion

Example 2 shows how the linear expansion in Remark 1 is ineffective even in very simple problem instances. The maximum likelihood bilinear model presented by Schäfer and Hüllermeier [26] is related to our model defined in Sect. 2 with the basis performing the Kronecker (tensor) product $\boldsymbol{\phi}(x, d) = x \otimes d$. Their results show such an expansion produces a competitive model, but falls behind in comparison to their non-linear model.

To model non-linear interactions between theorems and strategies, we use a *Gaussian kernel* for the basis expansion.

Definition 3 (Gaussian Kernel). *A Gaussian kernel κ is defined by*

$$\kappa(\boldsymbol{y}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{y} - \boldsymbol{z}\|^2}{2\sigma^2}\right), \quad \textit{for } \sigma > 0.$$

The Gaussian kernel $\kappa(\boldsymbol{y}, \boldsymbol{z})$ effectively represents the inner product of $\boldsymbol{y}$ and $\boldsymbol{z}$ in a Hilbert space whose bandwidth is controlled by σ. Smaller values of σ correspond to a higher bandwidth, more flexible, inner product space. Larger values of σ will reduce the kernel to a constant function, as detailed in [30]. For our ranking model, we must tune σ to balance between over-fitting and under-performance.

We use the Gaussian kernel for basis expansion by setting

$$\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{d}) = \left[\kappa\big([\boldsymbol{x}^\mathsf{T}, \boldsymbol{d}^\mathsf{T}]^\mathsf{T}, \boldsymbol{c}^{(1)}\big), \ldots, \kappa\big([\boldsymbol{x}^\mathsf{T}, \boldsymbol{d}^\mathsf{T}]^\mathsf{T}, \boldsymbol{c}^{(C)}\big)\right]^\mathsf{T},$$

where $\{\boldsymbol{c}^{(i)}\}_{i=1}^{C}$ is a collection of *centres*. By choosing centres to be themselves composed of theorems $\boldsymbol{x}^{(\cdot)}$ and strategies $\boldsymbol{d}^{(\cdot)}$, such that $\boldsymbol{c}^{(\cdot)} = [\boldsymbol{x}^{(\cdot)\mathsf{T}}, \boldsymbol{d}^{(\cdot)\mathsf{T}}]^\mathsf{T}$, the basis expansion above represents each data item with a non-linear inner product against other known items.

To find the relevant subset of $\mathcal{D}$ from which centres should be formed, we follow the method described in the steps below.

1. Initially, we set the collection of centres to every possible centre. That is, for N theorems and M strategies, we produce a centre for every combination of the two, thereby producing $C = N \cdot M$ centres.
2. Next, we use $\boldsymbol{\phi}$ to expand every centre to produce the $C \times C$ matrix $\boldsymbol{\Gamma}$ such that
$$\boldsymbol{\Gamma}_{i,j} = \boldsymbol{\phi}(\boldsymbol{c}^{(i)})_j = \kappa(\boldsymbol{c}^{(i)}, \boldsymbol{c}^{(j)}).$$
3. Then, we generate a vector $\boldsymbol{\gamma}$ such that $\boldsymbol{\gamma}_i$ represents a *score* for centre $\boldsymbol{c}^{(i)}$. Since each centre is a combination of a theorem and a strategy, we set the score to signify how well the strategy performs for that theorem, as detailed in Remark 7 below.
4. Finally, we use Automatic Relevance Determination (ARD) [17] with $\boldsymbol{\Gamma}$ as input and $\boldsymbol{\gamma}$ as the response variable. The result is a weight assignment to each centre to signify its relevance. The highest absolute-weighted ς centres are chosen, where ς is a parameter which decides the total number of centres.

This method is inspired by the procedure used in Relevance Vector Machines [35] for a similar purpose.

Remark 7 (score). For a strategy that succeeds in proving a theorem, the score for the pair is the fraction of the time allocation left unconsumed by the prover. For an unsuccessful strategy-theorem combination, we set the score to a value close to zero.

Remark 8 (ς). The parameter ς is another tunable parameter which, in similar fashion to the parameter σ earlier in this section, controls the model complexity introduced by the basis expansion. Both variables must be tuned together.

7 Model Selection and Time Allocations

From Remark 8, ς and σ are hyperparameters that control the complexity introduced into our model through the Gaussian basis expansion; and Remark 4 introduces η, the hyperparameter that controls the strength of the prior. The final model is selected by tuning them. Tuning must aim to avoid overfitting to the training data; and to maximize, during testing, either the savings in proof-search time or the number of theorems proved. However, we do not have a closed-form expression relating these parameters to this aim, thus any combination of the parameters can be judged only by testing them.

In this work we have used *Bayesian optimization* [29] to optimize these hyperparameters. Bayesian optimization is a black-box parameter optimization method that attempts to search for a global optimum within the scope of a set resource budget. It models the optimization target as a user-specified *objective function*, which maps from the parameter space to a loss metric. This model of the objective function is constructed using *Gaussian Process* (GP) regression [22], using data generated by repeatedly testing the objective function.

Our specified objective function maps from the hyperparameters $(\varsigma, \sigma, \eta)$ to a loss metric ξ. We use cross-validation within the training data while calculating ξ to penalize hyperparameters that over-fit. Hyperparameters are tuned at training time only, after which they are fixed for subsequent testing. The final test set is never used for any hyperparameter optimization.

In the method presented thus far we are only permuting strategies with fixed time allocations to build a sequence for a strategy schedule. In this setting, the number of theorems proved cannot change, but the time taken to prove theorems can be reduced. Therefore, with this aim, a useful metric for ξ is the total time taken by the theorem prover to prove the theorems in the cross-validation test set.

However, we can take further advantage of the hyperparameter tuning phase to additionally tune the times allocated to each strategy, by treating these times as hyperparameters. Therefore, for each strategy $\boldsymbol{d}^{(i)}$ we create a hyperparameter $\nu^{(i)} \in (0, 1)$ which sets the proportion of the proof time allocated to that strategy. We can then optimize our model to maximize the number of theorems proved; a count of the remaining theorems is then a viable metric for ξ. Note that once the $\nu^{(\cdot)}$ are set, time allocation for $\boldsymbol{d}^{(i)}$ is fixed to $\nu^{(i)}$, irrespective of its order in the strategy schedule.

Remark 9. Our results include two types of experiment:

- one where the time allocations for each strategy are set to the defaults shipped with our reference theorem prover, and so we optimize for saving proof time; and
- another wherein we allocate time to each strategy during the hyperparameter tuning phase, and so we optimize for proving the maximum number of theorems.

8 Training Data and Feature Extraction

Our chosen theorem prover, iLeanCoP, is shipped with a fixed strategy schedule consisting of 5 strategies. It splits the allocated proof time across the first four strategies by 2%, 60%, 20% and 10%. However, only the first strategy is complete and therefore usually expected to take up its entire time allocation. The remaining strategies are incomplete, and may exit early on failure. Therefore, the fifth and final strategy, which we refer to as the fallback strategy, is allocated all the remaining time.

Emulating iLeanCop. We have constructed a dataset by attempting to prove every theorem in our problem library using each of these strategies individually. With this information, the result of any proof attempt can be calculated by emulating the behaviour of iLeanCoP. This is how we evaluate the predicted schedules—we emulate a proof attempt by iLeanCoP using that schedule for each theorem in the test set. For a faithful emulation of the fallback strategy, it is always attempted last, and therefore any new schedule is only a permutation of the first four strategies. Our experiments allocate a time of 600 s per theorem. The dataset is built to ensure that, within this proof time, any such strategy permutation can be emulated. We kept a timeout of 1200 s per strategy per theorem when building the dataset, which is more than sufficient for current experiments and gives us headroom for future experiments with longer proof times.

Strategy Features. Each strategy in iLeanCoP consists of a time allocation and parameter settings; the parameters are described by Otten [19]. We use a one-hot encoding feature representation for strategies based on the parameter setting as shown in Table 1. Another feature noting the completeness of each strategy is also shown. Another feature (not shown in the table) contains the time allocated to each strategy. Note the fallback strategy is used in prover emulation but not in the schedule prediction.

Table 1. Features of the four main strategies.

Strategy	Parameter					Completeness
	`def`	`scut`	`cut`	`comp(7)`	`conj`	
`def,scut,cut,comp(7)`	1	1	1	1	0	1
`def,scut,cut`	1	1	1	0	0	0
`conj,scut,cut`	0	1	1	0	1	0
`def,conj,cut`	1	0	1	0	1	0

Theorem Features. The TPTP problem library contains a large, comprehensive collection of theorems and is designed for testing automated theorem provers. The problems are taken from a range of domains such as Logic Calculi, Algebra, Software Verification, Biology and Philosophy, and presented in multiple logical forms. For iLeanCoP, we select the subset in first-order form, denoted there as `FOF`. In version 7.1.0, there are 8157 such problems covering 43 domains. Each problem consists of a set of formulae and a goal theorem. The problems are of varying sizes. For example, the problem named `HWV134+1` from the Hardware Verification domain contains 128975 formulae, whilst `SET703+4` from the Set Theory domain contains only 12.

We have constructed a dataset containing features extracted from the first-order logic problems in TPTP (see Appendix A). Here, we describe how those features were developed.

In deployment, a prover using our method to generate strategy schedules would have to extract features from the goal theorem at the beginning of a proof attempt. To minimize the computational overhead of feature extraction, in keeping with our goal noted in Remark 3, we use features that can be collected when the theorem is parsed by the prover. The collection of features developed in this work is based on the authors' prior experience, and later we will briefly examine the quality of each feature to discard the uninformative ones. We extract the following features, which are all considered candidates for the subsequent feature selection process.

Symbol Counts: A count of the logical connectives and quantifiers. We extract one feature per symbol by tracking lexical symbols encountered while parsing.

Quantifier Rank: The maximum depth of nesting of quantifiers.

Quantifier Count: A count of the number of quantifiers.

Mean and Maximum Function Arity: Obtained by keeping track of functions during parsing.

Number of Functions: A count of the number of functions.

Quantifier Alternations: A count of the number of times the quantifiers flip between the existential and universal. When calculated by examining only the sequence of lexical symbols, the count may be inaccurate. An accurate count is obtained by tracking negations during parsing while collecting quantifiers. We extract both as candidates.

Feature Selection and Pre-processing. We examine the degree of association between the individual theorem features described above and the speed with which the strategies solve each theorem; for this we use the Maximal Information Coefficient (MIC) measure [23]. For every theorem we calculate the score, as defined in Remark 7, averaged over all strategies. This score is paired with each feature to calculate its MIC. Most lexical symbols achieve an MIC close to zero. We selected the features with relatively high MIC for the presented work, and these are shown in Fig. 3.

The two features based on quantifier alternations are clearly correlated, but both meet the above criterion for selection. Correlations can also be expected between the other features. Furthermore, our features range over different scales. For example, the maximal function arity in TPTP averages 2, whereas the number of predicate symbols averages 2097. It is desirable to remove these correlations to alleviate any burden on the subsequent modelling phase, and to standardize the features to zero mean and unit variance to create a feature space with similar length-scales in all dimensions. The former is achieved by *decorrelation*, the latter by *standardization*, and both together by a *sphering transformation*.

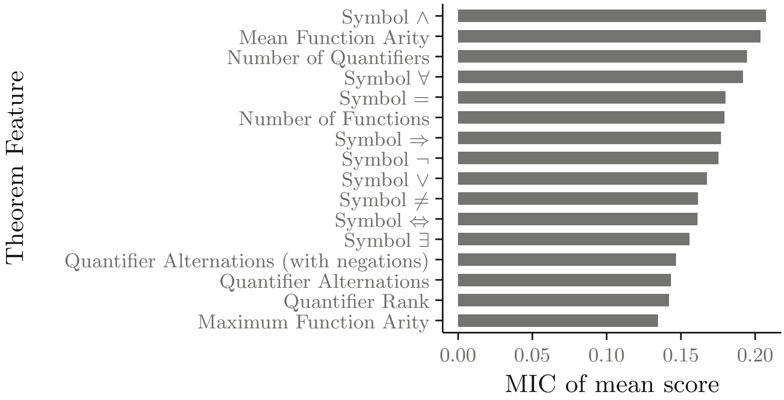


Fig. 3. MIC between selected features and scores.

We transform our extracted features as such using Zero-phase Component Analysis (ZCA), which ensures the transformed data is as close as possible to the original [6].

Coverage. As mentioned above, we run iLeanCoP on every first-order theorem in TPTP with each strategy allocated 1200 s. Although every theorem in intuitionistic logic also holds for classical logic, the converse does not hold. For that reason and because of the limitations of iLeanCoP, many theorems remain unproved by any strategy. We exclude these theorems from our experiments, leaving us with a data-set of 2240 theorems.

9 Experiments

We present two experiments in this work, as noted in Remark 9. In this section, we describe our experimental apparatus in detail.

As noted in Sect. 8, our data contains:

- $N = 2240$ theorems that are usable in our experiments;
- five strategies, of which $M = 4$ are used to build strategy schedules since one is a fallback strategy; and
- features $\boldsymbol{x}^{(i)}$ of theorems where $i \in [1, N]$ and features $\boldsymbol{d}^{(j)}$ of strategies where $j \in [1, M]$.

This data needs to be presented to our model for training in the form of $\mathcal{D} = \{(\boldsymbol{\pi}^{(i)}, \boldsymbol{x}^{(i)})\}_{i=1}^{N}$, as described in Sect. 3. Since the two experiments have slightly different goals, we specialize $\mathcal{D}$ according to each.

When aiming to predict schedules that minimize the time taken to prove theorems, a natural value for $\boldsymbol{\pi}^{(i)}$ is the index order that sorts strategies in increasing amounts of time taken to prove theorem i. However, some strategies may fail to prove theorem i within their time allocation. In that case, we consider the failed strategies equally bad and place them last in the ordering in $\boldsymbol{\pi}^{(i)}$. Furthermore, we create additional items $(\boldsymbol{\pi}'^{(i)}, \boldsymbol{x}^{(i)})$ in $\mathcal{D}$, by permuting the positions of the failed strategies in $\boldsymbol{\pi}^{(i)}$ to create multiple $\boldsymbol{\pi}'^{(i)}$.

When the goal is only to prove more theorems, the strategies that succeed are all considered equally ranked above the failed strategies. In this mode, the successful strategies are similarly permuted in the data, in addition to those that failed.

In each experiment, a random one-third of the N theorems are separated into a *holdout* test set $\dot{\boldsymbol{N}}$, leaving behind a training set $\ddot{\boldsymbol{N}}$. This training set is first used for hyperparameter tuning using BO. As explained in Sect. 7, each hyperparameter combination is tested with five-fold cross-validation within $\ddot{\boldsymbol{N}}$, to penalize instances that overfit to $\ddot{\boldsymbol{N}}$. This results in estimated optimum values for the hyperparameters. These are used to set the model, which is then trained on $\ddot{\boldsymbol{N}}$ and then finally evaluated on $\dot{\boldsymbol{N}}$. The whole process is repeated ten times with new random splits $\dot{\boldsymbol{N}}$ and $\ddot{\boldsymbol{N}}$ to create one set of ten results for that experiment.

10 Results

Each experiment, repeated ten times, is conducted in two phases: first, hyperparameter optimization; and second, model training and evaluation. The bounds on the search space in the first phase were always the same (see Appendix A). The holdout test set contained 747 theorems. A proof time of 600 s was emulated.

10.1 Experiment 1: Optimizing Proof Attempt Time

The results are shown in Fig. 4. The total prediction time for all 747 theorems, averaged across the trials, is 0.14 s.

The times across proof attempts are not normally distributed, for both the unmodified iLeanCoP schedule and the predicted ones, as confirmed by a Jarque-Bera test. Therefore, we used the right-tailed Wilcoxon signed-rank test for a pair-wise comparison of the times taken for each theorem by the original schedule in iLeanCoP versus the predicted schedules, resulting in a p-value of less

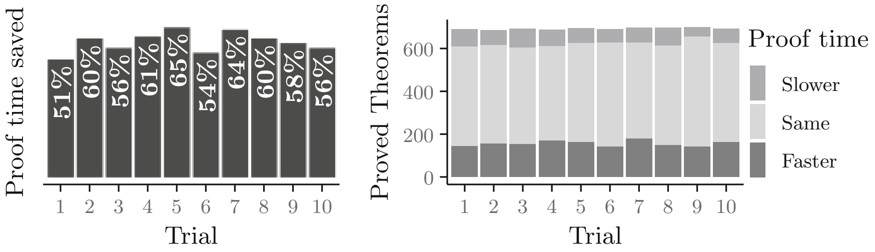


Fig. 4. Results of Experiment 1. Proof times are compared with precision 10^{-6}s.

than 10^{-6} in each trial, confirming the alternate hypothesis that the reduction in time taken to prove each theorem comes from a distribution with median greater than zero. This confirms that the time savings are statistically significant. Furthermore, we note from Fig. 4 a saving of more than 50% in the total proof-time in each trial.

10.2 Experiment 2: Proving More Theorems

We set our hyperparameter search to find time allocations for strategies. The resulting predicted schedules have gains and losses when compared to the original schedule, as shown in the four facets of Fig. 5. However, there is a consistent gain in the number of theorems proved and a gain of five theorems on average, evident from the mean values in (†) and (‡).

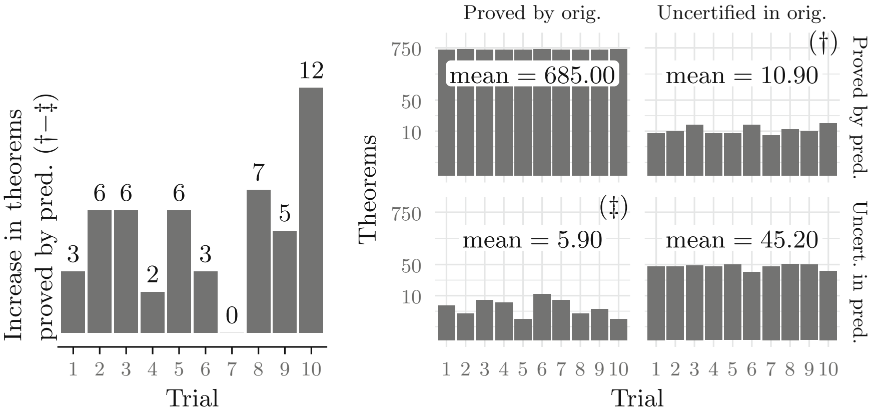


Fig. 5. Comparison of the proof attempts by the original (orig.) and predicted (pred.) schedules in Experiment 2. Theorems which are proved by pred. but could not be proved by orig. are counted in †, and the vice versa in ‡.

11 Related Work

Prior work on machine learning for *algorithm selection*, such as that introduced by Leyton-Brown et al. [13], is a precursor to our work. In that topic, the machine learning methods must perform the task of selecting a good algorithm from within a portfolio to solve the given problem instance. Typically, as was the case in the work by Leyton-Brown et al. [13], the learning methods predict the runtime of all algorithms, and then pick the fastest predicted one. This line of enquiry has been extended to select algorithms for SMT solvers—a recent example is MachSMT by Scott et al. [28]. The machine learning models in MachSMT are trained by considering all the portfolio members in pairs for each problem in the training set. This method is called *pairwise ranking*, which contrasts from our method, called *list-wise ranking*, in which we consider the full list of portfolio members all together.

In terms of the machine learning task, the work on scheduling solvers bears greater similarity to our presented work. In MedleySolver, for example, Pimpalkhare et al. [20] frame this task as a multi-armed bandit problem. They predict a sequence of solvers as well as the time allocation for each to generate schedules for the goal problems. MedleySolver is able to solve more problems than any individual solver would on its own.

With an approach that contrasts with ours, Hůla et al. [10] have made use of Graph Neural Networks (GNNs) for solver scheduling. They produce a regression model to predict, for the given problem, the runtime of all the solvers; which is used as the key to sort the solvers in increasing order of predicted runtime to build a schedule. This is an example of *point-wise ranking*. The authors use GNNs to automatically discover features for machine learning. They combine this feature extraction with training of the regression model. They achieve an increase in the number of problems solved as well as a reduction in the total proof time. Meanwhile, our use of manual feature engineering combined with statistical methods for selection and normalization has certain advantages. For one, we can analyse our features and derive a subjective interpretation of their efficacy. Additionally, our features effectively impart our domain knowledge onto the model. Such domain knowledge may not be available in the data itself. Manual feature engineering such as ours can be combined with automatic feature extraction to reap the benefits of both.

12 Conclusions

We have presented a method to specialize, for the given goal theorem, the sequence of strategies in the schedule used in each proof attempt. A Bayesian machine learning model is trained in this method using data generated by testing the prover of interest. When evaluated with the iLeanCoP prover using the TPTP library as a benchmark, our results show a significant reduction in the time taken to prove theorems. For theorems that are successfully proved, the average time saving is above 50%. The prediction time is on average low enough to have a negligible impact on the resources subtracted from the proof search itself.

We also extend this method to optimize time allocations to each strategy. In this setting, our results show a notable increase in the number of theorems proved.

This work shows, by example, that Bayesian machine learning models designed specifically to augment heuristics in theorem provers, with detailed consideration of the computational compromises required in this setting, can deliver substantial improvements.

Acknowledgments. Initial investigations for this work were co-supervised by Prof. Mateja Jamnik, Computer Laboratory, University of Cambridge, UK.

This work was supported by: the UK Engineering and Physical Sciences Research Council (EPSRC) through a Doctoral Training studentship, award reference 1788755; and the ERC Advanced Grant ALEXANDRIA (Project GA 742178). For the purpose of open access, the author has applied a Creative Commons Attribution (CC-BY-4.0) licence to any Author Accepted Manuscript version arising.

Computations for this work was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service, provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/P020259/1), and DiRAC funding from the Science and Technology Facilities Council.

A Implementation, Code and Data

This work is implemented primarily in Matlab [36]. All experiments can be reproduced using the code, data and instructions available at [15]. The hyperparameter search space in all experiments was restricted to $\varsigma \in [10, 300]$, $\sigma \in [0.01, 100.0]$ and $\eta \in [1, 100]$.

References

1. Balunovic, M., Bielik, P., Vechev, M.T.: Learning to solve SMT formulas. In: Annual Conference on Neural Information Processing Systems, pp. 10338–10349 (2018)
2. Barber, D.: Bayesian Reasoning and Machine Learning. Cambridge University Press, Cambridge (2012)
3. Bridge, J.P., Holden, S.B., Paulson, L.C.: Machine learning for first-order theorem proving. J. Autom. Reas. **53**(2), 141–172 (2014). https://doi.org/10.1007/s10817-014-9301-5
4. Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to rank: from pairwise approach to listwise approach. In: International Conference on Machine Learning, pp. 129–136. Association for Computing Machinery (2007)
5. Cheng, W., Dembczynski, K., Hüllermeier, E.: Label ranking methods based on the Plackett-Luce model. In: International Conference on Machine Learning, pp. 215–222. Omnipress (2010)
6. Duboue, P.: The Art of Feature Engineering: Essentials for Machine Learning. Cambridge University Press, Cambridge (2020)
7. Dummett, M.: Elements of Intuitionism, 2nd edn. Clarendon, Oxford (2000)
8. Guiver, J., Snelson, E.: Bayesian inference for Plackett-Luce ranking models. In: International Conference on Machine Learning, pp. 377–384. Association for Computing Machinery (2009)
9. Hales, T., et al.: A formal proof of the Kepler conjecture. In: Forum of Mathematics, Pi, vol. 5 (2017)
10. Hůla, J., Mojžíšek, D., Janota, M.: Graph neural networks for scheduling of SMT solvers. In: International Conference on Tools with Artificial Intelligence, pp. 447–451 (2021)
11. Johnson, S.R., Henderson, D.A., Boys, R.J.: On Bayesian inference for the Extended Plackett-Luce model (2020). arXiv:2002.05953

12. Kaliszyk, C., Urban, J., Vyskočil, J.: Machine learner for automated reasoning 0.4 and 0.5 (2014). arXiv:1402.2359
13. Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., Shoham, Y.: A portfolio approach to algorithm selection. In: International Joint Conference on Artificial Intelligence, pp. 1542–1543. Morgan Kaufmann Publishers Inc. (2003)
14. Luce, R.D.: Individual Choice Behavior: A Theoretical Analysis. Wiley, Hoboken (1959)
15. Mangla, C.: BRASS (2022). https://doi.org/10.5281/zenodo.6028568
16. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT press, Cambridge (2012)
17. Neal, R.M.: Bayesian Learning for Neural Networks. Springer, New York (1996). https://doi.org/10.1007/978-1-4612-0745-0
18. Otten, J.: Clausal connection-based theorem proving in intuitionistic first-order logic. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS (LNAI), vol. 3702, pp. 245–261. Springer, Heidelberg (2005). https://doi.org/10.1007/11554554_19
19. Otten, J.: leanCoP 2.0 and ileanCoP 1.2: high performance lean theorem proving in classical and intuitionistic logic (system descriptions). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 283–291. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71070-7_23
20. Pimpalkhare, N., Mora, F., Polgreen, E., Seshia, S.A.: MedleySolver: online SMT algorithm selection. In: Li, C.-M., Manyà, F. (eds.) SAT 2021. LNCS, vol. 12831, pp. 453–470. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-80223-3_31
21. Plackett, R.L.: The analysis of permutations. J. Roy. Stat. Soc. Ser. C (Appl. Stat.) **24**(2), 193–202 (1975)
22. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning, MIT Press, Cambridge (2006)
23. Reshef, D.N., et al.: Detecting novel associations in large data sets. Science **334**(6062), 1518–1524 (2011)
24. Roberts, G.O., Rosenthal, J.S.: Optimal scaling for various Metropolis-Hastings algorithms. Stat. Sci. **16**(4), 351–367 (2001)
25. Rossi, P.E.: Bayesian Statistics and Marketing. Wiley, Hoboken (2006)
26. Schäfer, D., Hüllermeier, E.: Dyad ranking using Plackett-Luce models based on joint feature representations. Mach. Learn. **107**(5), 903–941 (2018)
27. Schulz, S.: E - a Brainiac Theorem Prover. AI Commun. **15**(23), 111–126 (2002)
28. Scott, J., Niemetz, A., Preiner, M., Nejati, S., Ganesh, V.: MachSMT: a machine learning-based algorithm selector for SMT solvers. In: TACAS 2021. LNCS, vol. 12652, pp. 303–325. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72013-1_16
29. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N.: Taking the human out of the loop: a review of Bayesian optimization. Proc. IEEE **104**(1), 148–175 (2015)
30. Shawe-Taylor, J.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
31. Silverthorn, B., Miikkulainen, R.: Latent class models for algorithm portfolio methods. In: AAAI Conference on Artificial Intelligence, pp. 167–172. AAAI Press (2010)
32. Sutcliffe, G.: The TPTP problem library and associated infrastructure: from CNF to TH0, TPTP v.6.4.0. J. Autom. Reason. **59**(4), 483–502 (2017)
33. Tammet, T.: Gandalf. J. Autom. Reason. **18**(2), 199–204 (1997)

34. Thornton, C.: Parity: the problem that won't go away. In: McCalla, G. (ed.) AI 1996. LNCS, vol. 1081, pp. 362–374. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61291-2_65
35. Tipping, M.E.: Sparse Bayesian learning and the relevance vector machine. J. Mach. Learn. Res. **1**, 211–244 (2001)
36. Trauth, M.H.: MATLAB® Recipes for Earth Sciences. Springer, Heidelberg (2021). https://doi.org/10.1007/3-540-27984-9
37. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: system description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006). https://doi.org/10.1007/11814771_26
38. Wasserman, L.: All of Statistics: A Concise Course in Statistical Inference. Springer, New York (2004). https://doi.org/10.1007/978-0-387-21736-9

A Framework for Approximate Generalization in Quantitative Theories

Temur Kutsia(✉) and Cleo Pau

RISC, Johannes Kepler University Linz, Linz, Austria
{kutsia,ipau}@risc.jku.at

Abstract. Anti-unification aims at computing generalizations for given terms, retaining their common structure and abstracting differences by variables. We study quantitative anti-unification where the notion of the common structure is relaxed into "proximal" up to the given degree with respect to the given fuzzy proximity relation. Proximal symbols may have different names and arities. We develop a generic set of rules for computing minimal complete sets of approximate generalizations and study their properties. Depending on the characterizations of proximities between symbols and the desired forms of solutions, these rules give rise to different versions of concrete algorithms.

Keywords: Generalization · Anti-unification · Quantiative theories · Fuzzy proximity relations

1 Introduction

Generalization problems play an important role in various areas of mathematics, computer science, and artificial intelligence. Anti-unification [12,14] is a logic-based method for computing generalizations. Being originally used for inductive and analogical reasoning, some recent applications include recursion scheme detection in functional programs [4], programming by examples in domain-specific languages [13], learning bug-fixing from software code repositories [3,15], automatic program repair [7], preventing bugs and misconfiguration in services [11], linguistic structure learning for chatbots [6], to name just a few.

In most of the existing theories where anti-unification is studied, the background knowledge is assumed to be precise. Therefore, those techniques are not suitable for reasoning with incomplete, imprecise information (which is very common in real-world communication), where the exact equality is replaced by its (quantitative) approximation. Fuzzy proximity and similarity relations are notable examples of such extensions. These kinds of quantitative theories have many useful applications, some most recent ones being related to artificial intelligence, program verification, probabilistic programming, or natural language processing. Many tasks arising in these areas require reasoning methods and computational tools that deal with quantitative information. For instance, approximate

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 578–596, 2022.
https://doi.org/10.1007/978-3-031-10769-6_34

inductive reasoning, reasoning and programming by analogy, similarity detection in programming language statements or in natural language texts could benefit from solving approximate generalization constraints, which is a theoretically interesting and challenging task. Investigations in this direction have been started only recently. In [1], the authors proposed an anti-unification algorithm for fuzzy similarity (reflexive, symmetric, min-transitive) relations, where mismatches are allowed not only in symbol names, but also in their arities (fully fuzzy signatures). The algorithm from [9] is designed for fuzzy proximity (i.e., reflexive and symmetric) relations with mismatches only in symbol names.

In this paper, we study approximate anti-unification from a more general perspective. The considered relations are fuzzy proximity relations. Proximal symbols may have different names and arities. We consider four different variants of relating arguments between different proximal symbols: unrestricted relations/functions, and correspondence (i.e. left- and right-total) relations/functions. A generic set of rules for computing minimal complete sets of generalizations is introduced and its termination, soundness and completeness properties are proved. From these rules, we obtain concrete algorithms that deal with different kinds of argument relations. We also show how the existing approximate anti-unification algorithms and their generalizations fit into this framework.

Organization: In Sect. 2 we introduce the notation and definitions. Section 3 is devoted to a technical notion of term set consistency and to an algorithm for computing elements of consistent sets of terms. It is used later in the main set of anti-unification rules, which are introduced and characterized in Sect. 4. The concrete algorithms obtained from those rules are also described in this section. In Sect. 5, we discuss complexity. Section 6 offers a high-level picture of the studied problems and concludes.

An extended version of this work can be found in the technical report [8].

2 Preliminaries

Proximity Relations. Given a set S, a mapping $\mathcal{R}$ from $S \times S$ to the real interval $[0, 1]$ is called a binary *fuzzy relation* on S. By fixing a number λ, $0 \leqslant \lambda \leqslant 1$, we can define the crisp (i.e., two-valued) counterpart of $\mathcal{R}$, named the λ-*cut* of $\mathcal{R}$, as $\mathcal{R}_\lambda := \{(s_1, s_2) \mid \mathcal{R}(s_1, s_2) \geqslant \lambda\}$. A fuzzy relation $\mathcal{R}$ on a set S is called a *proximity relation* if it is reflexive ($\mathcal{R}(s, s) = 1$ for all $s \in S$) and symmetric ($\mathcal{R}(s_1, s_2) = \mathcal{R}(s_2, s_1)$ for all $s_1, s_2 \in S$). A T-norm $\wedge$ is an associative, commutative, non-decreasing binary operation on $[0, 1]$ with 1 as the unit element. We take minimum in the role of T-norm.

Terms and Substitutions. We consider a first-order alphabet consisting of a set of fixed arity function symbols $\mathcal{F}$ and a set of variables $\mathcal{V}$, which includes a special symbol _ (the anonymous variable). The set of *named* (i.e., non-anonymous) variables $\mathcal{V} \backslash \{\_\}$ is denoted by $\mathcal{V}^{\mathrm{N}}$. When the set of variables is not explicitly specified, we mean $\mathcal{V}$. The set of terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$ over $\mathcal{F}$ and $\mathcal{V}$ is defined in the standard way: $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ iff t is defined by the grammar $t := x \mid f(t_1, \dots, t_n)$, where $x \in \mathcal{V}$ and $f \in \mathcal{F}$ is an n-ary symbol with $n \geqslant 0$. Terms over $\mathcal{T}(\mathcal{F}, \mathcal{V}^{\mathrm{N}})$ are defined similarly except that all variables are taken from $\mathcal{V}^{\mathrm{N}}$.

We denote arbitrary function symbols by f, g, h, constants by a, b, c, variables by x, y, z, v, and terms by s, t, r. The *head* of a term is defined as $\mathsf{head}(x) := x$ and $\mathsf{head}(f(t_1, \dots, t_n)) := f$. For a term t, we denote with $\mathcal{V}(t)$ (resp. by $\mathcal{V}^{\mathrm{N}}(t)$) the set of all variables (resp. all named variables) appearing in t. A term is called *linear* if no named variable occurs in it more than once.

The deanonymization operation deanon replaces each occurrence of the anonymous variable in a term by a fresh variable. For instance, $\mathsf{deanon}(f(\_, x, g(\_))) = f(y', x, g(y'')))$, where y' and y'' are fresh. Hence, $\mathsf{deanon}(t) \in \mathcal{T}(\mathcal{F}, \mathcal{V}^{\mathrm{N}})$ is unique up to variable renaming for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. $\mathsf{deanon}(t)$ is linear iff t is linear.

The notions of *term depth*, *term size* and a *position in a term* are defined in the standard way, see, e.g. [2]. By $t|_p$ we denote the subterm of t at position p and by $t[s]_p$ a term that is obtained from t by replacing the subterm at position p by the term s.

A *substitution* is a mapping from $\mathcal{V}^{\mathrm{N}}$ to $\mathcal{T}(\mathcal{F}, \mathcal{V}^{\mathrm{N}})$ (i.e., without anonymous variables), which is the identity almost everywhere. We use the Greek letters $\sigma, \vartheta, \varphi$ to denote substitutions, except for the identity substitution which is written as Id. We represent substitutions with the usual set notation. *Application* of a substitution σ to a term t, denoted by $t\sigma$, is defined as $\_\sigma := \_$, $x\sigma := \sigma(x)$, $f(t_1, \dots, t_n)\sigma := f(t_1\sigma, \dots, t_n\sigma)$. Substitution *composition* is defined as a composition of mappings. We write $\sigma\vartheta$ for the composition of σ with ϑ.

Argument Relations and Mappings. Given two sets $N = \{1, \dots, n\}$ and $M = \{1, \dots, m\}$, a binary *argument relation* over $N \times M$ is a (possibly empty) subset of $N \times M$. We denote argument relations by ρ. An argument relation $\rho \subseteq N \times M$ is (i) *left-total* if for all $i \in N$ there exists $j \in M$ such that $(i, j) \in \rho$; (ii) *right-total* if for all $j \in M$ there exists $i \in N$ such that $(i, j) \in \rho$. *Correspondence relations* are those that are both left- and right-total.

An *argument mapping* is an argument relation that is a partial injective function. In other words, an argument mapping π from $N = \{1, \dots, n\}$ to $M = \{1, \dots, m\}$ is a function $\pi : I_n \mapsto I_m$, where $I_n \subseteq N$, $I_m \subseteq M$ and $|I_n| = |I_m|$. Note that it can be also the empty mapping: $\pi : \emptyset \mapsto \emptyset$. The inverse of an argument mapping is again an argument mapping.

Given a proximity relation $\mathcal{R}$ over $\mathcal{F}$, we assume that for each pair of function symbols f and g with $\mathcal{R}(f, g) = \alpha > 0$, where f is n-ary and g is m-ary, there is also given an argument relation ρ over $\{1, \dots, n\} \times \{1, \dots, m\}$. We use the notation $f \sim^{\rho}_{\mathcal{R},\alpha} g$. These argument relations should satisfy the following conditions: ρ is the empty relation if f or g is a constant; ρ is the identity if $f = g$; $f \sim^{\rho}_{\mathcal{R},\alpha} g$ iff $g \sim^{\rho^{-1}}_{\mathcal{R},\alpha} f$, where ρ^{-1} is the inverse of ρ.

Example 1. Assume that we have four different versions of defining the notion of author (e.g., originated from four different knowledge bases) *author*$_1$(*first*-name, *middle*-*initial*, *last*-name), *author*$_2$(*first*-name, *last*-name), *author*$_3$(*last*-name, *first*-name, *middle*-*initial*), and *author*$_4$(*full*-name). One could define the argument relations/mappings between these function symbols e.g., as follows:

$$\begin{array}{ll} author_1 \sim_{\mathcal{R},0.7}^{\{(1,1),(3,2)\}} author_2, & author_1 \sim_{\mathcal{R},0.9}^{\{(3,1),(1,2),(2,3)\}} author_3, \\ author_1 \sim_{\mathcal{R},0.5}^{\{(1,1),(3,1)\}} author_4, & author_2 \sim_{\mathcal{R},0.7}^{\{(1,2),(2,1)\}} author_3, \\ author_2 \sim_{\mathcal{R},0.5}^{\{(1,1),(2,1)\}} author_4, & author_3 \sim_{\mathcal{R},0.5}^{\{(1,1),(2,1)\}} author_4. \end{array}$$

Proximity Relations over Terms. Each proximity relation $\mathcal{R}$ in this paper is defined on $\mathcal{F} \cup \mathcal{V}$ such that $\mathcal{R}(f,x) = 0$ for all $f \in \mathcal{F}$ and $x \in \mathcal{V}$, and $\mathcal{R}(x,y) = 0$ for all $x \neq y$, $x, y \in \mathcal{V}$. We assume that $\mathcal{R}$ is *strict*: for all $w_1, w_2 \in \mathcal{F} \cup \mathcal{V}$, if $\mathcal{R}(w_1, w_2) = 1$, then $w_1 = w_2$. Yet another assumption is that for each $f \in \mathcal{F}$, its $(\mathcal{R}, \lambda)$-proximity class $\{g \mid \mathcal{R}(f,g) \geqslant \lambda\}$ is *finite* for any $\mathcal{R}$ and λ.

We extend such an $\mathcal{R}$ to terms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ as follows:

(a) $\mathcal{R}(t,s) := 0$ if $\mathcal{R}(\mathsf{head}(s), \mathsf{head}(t)) = 0$;
(b) $\mathcal{R}(t,s) := 1$ if $t = s$ and $t, s \in \mathcal{V}$;
(c) $\mathcal{R}(t,s) := \mathcal{R}(f,g) \wedge \mathcal{R}(t_{i_1}, s_{j_1}) \wedge \cdots \wedge \mathcal{R}(t_{i_k}, s_{j_k})$, if $t = f(t_1, \ldots, t_n)$, $s = g(s_1, \ldots, s_m)$, $f \sim_{\mathcal{R},\lambda}^{\rho} g$, and $\rho = \{(i_1, j_1), \ldots, (i_k, j_k)\}$.

If $\mathcal{R}(t,s) \geqslant \lambda$, we write $t \simeq_{\mathcal{R},\lambda} s$. When $\lambda = 1$, the relation $\simeq_{\mathcal{R},\lambda}$ does not depend on $\mathcal{R}$ due to strictness of the latter and is just the syntactic equality $=$.

The $(\mathcal{R}, \lambda)$-*proximity class* of a term t is $\mathbf{pc}_{\mathcal{R},\lambda}(t) := \{s \mid s \simeq_{\mathcal{R},\lambda} t\}$.

Generalizations. Given $\mathcal{R}$ and λ, a term r is an $(\mathcal{R}, \lambda)$-*generalization* of (alternatively, $(\mathcal{R}, \lambda)$-*more general than*) a term t, written as $r \precsim_{\mathcal{R},\lambda} t$, if there exists a substitution σ such that $\mathsf{deanon}(r)\sigma \simeq_{\mathcal{R},\lambda} \mathsf{deanon}(t)$. The strict part of $\precsim_{\mathcal{R},\lambda}$ is denoted by $\prec_{\mathcal{R},\lambda}$, i.e., $r \prec_{\mathcal{R},\lambda} t$ if $r \precsim_{\mathcal{R},\lambda} t$ and not $t \precsim_{\mathcal{R},\lambda} r$.

Example 2. Given a proximity relation $\mathcal{R}$, a cut value λ, constants $a \sim_{\mathcal{R},\alpha_1}^{\varnothing} b$ and $b \sim_{\mathcal{R},\alpha_2}^{\varnothing} c$, binary function symbols f and h, and a unary function symbol g such that $h \sim_{\mathcal{R},\alpha_3}^{\{(1,1),(1,2)\}} f$ and $h \sim_{\mathcal{R},\alpha_4}^{\{(1,1)\}} g$ with $\alpha_i \geqslant \lambda$, $1 \leqslant i \leqslant 4$, we have

- $h(x, \_) \precsim_{\mathcal{R},\lambda} h(a,x)$, because $h(x,x')\{x \mapsto a, x' \mapsto x\} = h(a,x) \simeq_{\mathcal{R},\lambda} h(a,x)$.
- $h(x, \_) \precsim_{\mathcal{R},\lambda} h(\_,x)$, because $h(x,x')\{x \mapsto y', x' \mapsto x\} = h(y',x) \simeq_{\mathcal{R},\lambda} h(y',x)$.
- $h(x,x) \not\precsim_{\mathcal{R},\lambda} h(\_,x)$, because $h(x,x) \not\precsim_{\mathcal{R},\lambda} h(y',x)$.
- $h(x, \_) \precsim_{\mathcal{R},\lambda} f(a,c)$, because $h(x,x')\{x \mapsto b\} = h(b,x') \simeq_{\mathcal{R},\lambda} f(a,c)$.
- $h(x, \_) \precsim_{\mathcal{R},\lambda} g(c)$, because $h(x,x')\{x \mapsto c\} = h(c,x') \simeq_{\mathcal{R},\lambda} g(c)$.

The notion of *syntactic generalization* of a term is a special case of $(\mathcal{R}, \lambda)$-generalization for $\lambda = 1$. We write $r \precsim t$ to indicate that r is a syntactic generalization of t. Its strict part is denoted by $\prec$.

Since $\mathcal{R}$ is strict, $r \precsim t$ is equivalent to $\mathsf{deanon}(r)\sigma = \mathsf{deanon}(t)$ for some σ (note the syntactic equality here).

Theorem 1. *If $r \lesssim t$ and $t \lesssim_{\mathcal{R},\lambda} s$, then $r \lesssim_{\mathcal{R},\lambda} s$.*

Proof. $r \lesssim t$ implies $\mathsf{deanon}(r)\sigma = \mathsf{deanon}(t)$ for some σ, while from $t \lesssim_{\mathcal{R},\lambda} s$ we have $\mathsf{deanon}(t)\vartheta \simeq_{\mathcal{R},\lambda} \mathsf{deanon}(s)$ for some ϑ. Then $\mathsf{deanon}(r)\sigma\vartheta \simeq_{\mathcal{R},\lambda} \mathsf{deanon}(s)$, which implies $r \lesssim_{\mathcal{R},\lambda} s$. □

Note that $r \lesssim_{\mathcal{R},\lambda} t$ and $t \lesssim_{\mathcal{R},\lambda} s$, in general, do not imply $r \lesssim_{\mathcal{R},\lambda} s$ due to non-transitivity of $\simeq_{\mathcal{R},\lambda}$.

Definition 1 (Minimal complete set of $(\mathcal{R}, \lambda)$-generalizations). *Given $\mathcal{R}$, λ, t_1, and t_2, a set of terms T is a* complete set of $(\mathcal{R}, \lambda)$-generalizations *of t_1 and t_2 if*

(a) every $r \in T$ is an $(\mathcal{R}, \lambda)$-generalization of t_1 and t_2,
(b) if r' is an $(\mathcal{R}, \lambda)$-generalization of t_1 and t_2, then there exists $r \in T$ such that $r' \lesssim r$ (note that we use syntactic generalization here).

In addition, T is minimal, if it satisfies the following property:

(c) if $r, r' \in T$, $r \neq r'$, then neither $r \prec_{\mathcal{R},\lambda} r'$ nor $r' \prec_{\mathcal{R},\lambda} r$.

A minimal complete set of $(\mathcal{R}, \lambda)$-generalizations *($(\mathcal{R}, \lambda)$-mcsg) of two terms is unique modulo variable renaming. The elements of the $(\mathcal{R}, \lambda)$-mcsg of t_1 and t_2 are called least general $(\mathcal{R}, \lambda)$-generalizations ($(\mathcal{R}, \lambda)$-lggs) of t_1 and t_2.*

This definition directly extends to generalizations of finitely many terms.

The problem of computing an $(\mathcal{R}, \lambda)$-generalization of terms t and s is called the $(\mathcal{R}, \lambda)$*-anti-unification problem* of t and s. In anti-unification, the goal is to compute their least general $(\mathcal{R}, \lambda)$-generalization.

The precise formulation of the anti-unification problem would be the following: Given $\mathcal{R}$, λ, t_1, t_2, find an $(\mathcal{R}, \lambda)$-lgg r of t_1 and t_2, substitutions σ_1, σ_2, and the approximation degrees α_1, α_2 such that $\mathcal{R}(r\sigma_1, t_1) = \alpha_1$ and $\mathcal{R}(r\sigma_2, t_2) = \alpha_2$. A minimal complete algorithm to solve this problem would compute exactly the elements of $(\mathcal{R}, \lambda)$-mcsg of t_1 and t_2 together with their approximation degrees. However, as we see below, it is problematic to solve the problem in this form. Therefore, we will consider a slightly modified variant, taking into account anonymous variables in generalizations and relaxing bounds on their degrees.

We assume that the terms to be generalized are ground. It is not a restriction because we can treat variables as constants that are close only to themselves.

Recall that the proximity class of any alphabet symbol is finite. Also, the symbols are related to each other by finitely many argument relations. One may think that it leads to finite proximity classes of terms, but this is not the case. Consider, e.g., $\mathcal{R}$ and λ, where $h \simeq_{\mathcal{R},\lambda}^{\{(1,1)\}} f$ with binary h and unary f. Then the $(\mathcal{R}, \lambda)$-proximity class of $f(a)$ is infinite: $\{f(a)\} \cup \{h(a, t) \mid t \in \mathcal{T}(\mathcal{F}, \mathcal{V})\}$. Also, the $(\mathcal{R}, \lambda)$-mcsg for $f(a)$ and $f(b)$ is infinite: $\{f(x)\} \cup \{h(x, t) \mid t \in \mathcal{T}(\mathcal{F}, \varnothing)\}$.

Definition 2. *Given the terms $t_1, \ldots, t_n$, $n \geqslant 1$, a position p in a term r is called* irrelevant for $(\mathcal{R}, \lambda)$-generalizing *(resp.* for $(\mathcal{R}, \lambda)$-proximity to*) $t_1, \ldots, t_n$ if $r[s]_p \lesssim_{\mathcal{R},\lambda} t_i$ (resp. $r[s]_p \simeq_{\mathcal{R},\lambda} t_i$) for all $1 \leqslant i \leqslant n$ and for all terms s.*

We say that r is a relevant $(\mathcal{R},\lambda)$-generalization *(resp.* relevant $(\mathcal{R},\lambda)$-proximal term*) of $t_1,\dots,t_n$ if $r \lesssim_{\mathcal{R},\lambda} t_i$ (resp. $r \simeq_{\mathcal{R},\lambda} t_i$) for all $1 \leqslant i \leqslant n$ and $r|_p = \_$ for all positions p in r that is irrelevant for generalizing (resp. for proximity to) $t_1,\dots,t_n$. The $(\mathcal{R},\lambda)$*-relevant proximity class *of t is*

$$\mathbf{rpc}_{\mathcal{R},\lambda}(t) := \{s \mid s \textit{ is a relevant } (\mathcal{R},\lambda)\textit{-proximal term of } t\}.$$

In the example above, position 2 in $h(x,t)$ is irrelevant for generalizing $f(a)$ and $f(b)$, and $h(x,\_)$ is one of their relevant generalizations. Note that $f(x)$ is also a relevant generalization of $f(a)$ and $f(b)$, since it contains no irrelevant positions. More general generalizations like, e.g., x, are relevant as well. Similarly, position 2 in $h(a,t)$ is irrelevant for proximity to $f(a)$ and $\mathbf{rpc}_{\mathcal{R},\lambda}(f(a)) = \{f(a), h(a,\_)\}$. Generally, $\mathbf{rpc}_{\mathcal{R},\lambda}(t)$ is finite for any t due to the finiteness of proximity classes of symbols and argument relations mentioned above.

Definition 3 (Minimal complete set of relevant $(\mathcal{R},\lambda)$-generalizations). *Given $\mathcal{R}$, λ, t_1, and t_2, a set of terms T is a* complete set of relevant $(\mathcal{R},\lambda)$-generalizations *of t_1 and t_2 if*

(a) every element of T is a relevant $(\mathcal{R},\lambda)$-generalization of t_1 and t_2, and
(b) if r is a relevant $(\mathcal{R},\lambda)$-generalization of t_1 and t_2, then there exists $r' \in T$ such that $r \lesssim r'$.

The minimality property is defined as in Definition 1.

This definition directly extends to relevant generalizations of finitely many terms. We use $(\mathcal{R},\lambda)$-mcsrg as an abbreviation for minimal complete set of relevant $(\mathcal{R},\lambda)$-generalization. Like relevant proximity classes, mcsrg's are also finite.

Lemma 1. *For given $\mathcal{R}$ and λ, if all argument relations are correspondence relations, then $(\mathcal{R},\lambda)$-mcsg's and $(\mathcal{R},\lambda)$-proximity classes for all terms are finite.*

Proof. Under correspondence relations no term contains an irrelevant position for generalization or for proximity. □

Hence, for correspondence relations the notions of mcsg and mcsrg coincide, as well as the notions of proximity class and relevant proximity class.

For a term r, we define its *linearized version* $\mathsf{lin}(r)$ as a term obtained from r by replacing each occurrence of a named variable in r by a fresh one. For instance, $\mathsf{lin}(f(x,\_,g(y,x,a),b)) = f(x',\_,g(y',x'',a),b)$, where x', x'', y' are fresh variables. Linearized versions of terms are unique modulo variable renaming.

Definition 4 (Generalization degree upper bound). *Given two terms r and t, a proximity relation $\mathcal{R}$, and a λ-cut, the $(\mathcal{R},\lambda)$*-generalization degree upper bound of r and t, *denoted by $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t)$, is defined as follows:*

Let $\alpha := \max\{\mathcal{R}(\mathsf{lin}(r)\sigma, t) \mid \sigma \text{ is a substitution}\}$. Then $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t)$ is α if $\alpha \geqslant \lambda$, and 0 otherwise.

Intuitively, $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t) = \alpha$ means that no instance of r can get closer than α to t in $\mathcal{R}$. From the definition it follows that if $r \lesssim_{\mathcal{R},\lambda} t$, then $0 < \lambda \leqslant \mathsf{gdub}_{\mathcal{R},\lambda}(r,t) \leq 1$ and if $r \not\lesssim_{\mathcal{R},\lambda} t$, then $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t) = 0$.

The upper bound computed by gdub is more relaxed than it would be if the linearization function were not used, but this is what we will be able to compute in our algorithms later.

Example 3. Let $\mathcal{R}(a,b) = 0.6$, $\mathcal{R}(b,c) = 0.7$, and $\lambda = 0.5$. Then $\mathsf{gdub}_{\mathcal{R},\lambda}(f(x,b), f(a,c)) = 0.7$ and $\mathsf{gdub}_{\mathcal{R},\lambda}(f(x,x), f(a,c)) = \mathsf{gdub}_{\mathcal{R},\lambda}(f(x,y), f(a,c)) = 1$.

It is not difficult to see that if $r\sigma \simeq_{\mathcal{R},\lambda} t$, then $\mathcal{R}(r\sigma,t) \leqslant \mathsf{gdub}_{\mathcal{R},\lambda}(r,t)$. In Example 3, for $\sigma = \{x \mapsto b\}$ we have $\mathcal{R}(f(x,x)\sigma, f(a,c)) = \mathcal{R}(f(b,b), f(a,c)) = 0.6 < \mathsf{gdub}_{\mathcal{R},\lambda}(f(x,x), f(a,c)) = 1$.

We compute $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t)$ as follows: If r is a variable, then $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t) = 1$. Otherwise, if $\mathsf{head}(r) \sim^{\rho}_{\mathcal{R},\beta} \mathsf{head}(t)$, then $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t) = \beta \wedge \bigwedge_{(i,j)\in\rho} \mathsf{gdub}_{\mathcal{R},\lambda}(r|_i, t|_j)$. Otherwise, $\mathsf{gdub}_{\mathcal{R},\lambda}(r,t) = 0$.

3 Term Set Consistency

The notion of term set consistency plays an important role in the computation of proximal generalizations. Intuitively, a set of terms is $(\mathcal{R},\lambda)$-consistent if all the terms in the set have a common $(\mathcal{R},\lambda)$-proximal term. In this section, we discuss this notion and the corresponding algorithms.

Definition 5 (Consistent set of terms). *A finite set of terms T is $(\mathcal{R},\lambda)$-*consistent *if there exists a term s such that $s \simeq_{\mathcal{R},\lambda} t$ for all $t \in T$.*

$(\mathcal{R},\lambda)$-consistency of a finite term set T is equivalent to $\bigcap_{t\in T} \mathbf{pc}_{\mathcal{R},\lambda}(t) \neq \emptyset$, but we cannot use this property to decide consistency, since proximity classes of terms can be infinite (when the argument relations are not restricted). For this reason, we introduce the operation $\sqcap$ on terms as follows: (i) $t \sqcap \_ = \_ \sqcap t = t$, (ii) $f(t_1,\dots,t_n) \sqcap f(s_1,\dots,s_n) = f(t_1 \sqcap s_1,\dots,t_n \sqcap s_n)$, $n \geqslant 0$. Obviously, $\sqcap$ is associative (A), commutative (C), idempotent (I), and has _ as its unit element (U). It can be extended to sets of terms: $T_1 \sqcap T_2 := \{t_1 \sqcap t_2 \mid t_1 \in T_1, t_2 \in T_2\}$. It is easy to see that $\sqcap$ on sets also satisfies the ACIU properties with the set $\{\_\}$ playing the role of the unit element.

Lemma 2. *A finite set of terms T is $(\mathcal{R},\lambda)$-consistent iff $\bigsqcap_{t\in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t) \neq \emptyset$.*

Proof. ($\Rightarrow$) If $s \simeq_{\mathcal{R},\lambda} t$ for all $t \in T$, then $s_t \in \mathbf{rpc}_{\mathcal{R},\lambda}(t)$, where s_t is obtained from s by replacing all subterms that are irrelevant for its $(\mathcal{R},\lambda)$-proximity to t by _. Assume $T = \{t_1,\dots,t_n\}$. Then $s_{t_1} \sqcap \cdots \sqcap s_{t_n} \in \bigsqcap_{t\in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$.

($\Leftarrow$) Obvious, since $s \simeq_{\mathcal{R},\lambda} t$ for $s \in \bigsqcap_{t\in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$ and for all $t \in T$. □

Now we design an algorithm $\mathfrak{C}$ that computes $\bigsqcap_{t\in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$ without actually computing $\mathbf{rpc}_{\mathcal{R},\lambda}(t)$ for each $t \in T$. A special version of the algorithm can be used to decide the $(\mathcal{R},\lambda)$-consistency of T.

The algorithm is rule-based. The rules work on states, that are pairs $\mathbf{I}; s$, where s is a term and $\mathbf{I}$ is a finite set of expressions of the form x in T, where T is a finite set of terms. $\mathcal{R}$ and λ are given. There are two rules ($\uplus$ stands for disjoint union):

Rem: **Removing the empty set**
$\{x \textsf{ in } \varnothing\} \uplus \mathbf{I}; s \Longrightarrow \mathbf{I}; s\{x \mapsto \_\}$.

Red: **Reduce a set to new sets**
$\{x \textsf{ in } \{t_1, \dots, t_m\}\} \uplus \mathbf{I}; s \Longrightarrow \{y_1 \textsf{ in } T_1, \dots, y_n \textsf{ in } T_n\} \cup \mathbf{I}; s\{x \mapsto h(y_1, \dots, y_n)\}$, where $m \geqslant 1$, h is an n-ary function symbol such that $h \sim^{\rho_k}_{\mathcal{R},\gamma_k} \textsf{head}(t_k)$ with $\gamma_k \geqslant \lambda$ for all $1 \leqslant k \leqslant m$, and $T_i := \{t_k|_j \mid (i,j) \in \rho_k, 1 \leqslant k \leqslant m\}$, $1 \leqslant i \leqslant n$, is the set of all those arguments of the terms $t_1, \dots, t_m$ that are supposed to be $(\mathcal{R}, \lambda)$-proximal to the i's argument of h.

To compute $\bigsqcap_{t \in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$, $\mathfrak{C}$ starts with $\{x \textsf{ in } T\}; x$ and applies the rules as long as possible. Red causes branching. A state of the form $\varnothing; s$ is called a success state. A failure state has the form $\mathbf{I}; s$, to which no rule applies and $\mathbf{I} \neq \varnothing$. In the full derivation tree, each leaf is a either success or a failure state.

Example 4. Assume a, b, c are constants, g, f, h are function symbols with the arities respectively 1, 2, and 3. Let λ be given and $\mathcal{R}$ be defined so that $\mathcal{R}(a,b) \geqslant \lambda$, $\mathcal{R}(b,c) \geqslant \lambda$, $h \sim^{\{(1,1),(1,2)\}}_{\mathcal{R},\beta} f$, $h \sim^{\{(2,1)\}}_{\mathcal{R},\gamma} g$ with $\beta \geqslant \lambda$ and $\gamma \geqslant \lambda$. Then

$$\begin{aligned} \mathbf{rpc}_{\mathcal{R},\lambda}(f(a,c)) &= \{f(a,c),\ f(b,c),\ f(a,b),\ f(b,b),\ h(b,\_,\_)\}, \\ \mathbf{rpc}_{\mathcal{R},\lambda}(g(a)) &= \{g(a),\ g(b), h(\_,a,\_), h(\_,b,\_)\}, \end{aligned}$$

and $\mathbf{rpc}_{\mathcal{R},\lambda}(f(a,c)) \sqcap \mathbf{rpc}_{\mathcal{R},\lambda}(g(a)) = \{h(b,a,\_), h(b,b,\_)\}$. We show how to compute this set with $\mathfrak{C}$: $\{x \textsf{ in } \{f(a,c), g(a)\}\};\ x \Longrightarrow_{\textsf{Red}} \{y_1 \textsf{ in } \{a,c\}, y_2 : \{a\}, y_3 \textsf{ in } \varnothing\};\ h(y_1,y_2,y_3) \Longrightarrow_{\textsf{Rem}} \{y_1 \textsf{ in } \{a,c\},\ y_2 : \{a\}\};\ h(y_1,y_2,\_) \Longrightarrow_{\textsf{Red}} \{y_2 \textsf{ in } \{a\}\}; h(b,y_2,\_)$. Here we have two ways to apply Red to the last state, leading to two elements of $\mathbf{rpc}_{\mathcal{R},\lambda}(f(a,c)) \sqcap \mathbf{rpc}_{\mathcal{R},\lambda}(g(a))$: $h(b,a,\_)$ and $h(b,b,\_)$.

Theorem 2. *Given a finite set of terms T, the algorithm $\mathfrak{C}$ always terminates starting from the state $\{x \textsf{ in } T\}; x$ (where x is a fresh variable). If S is the set of success states produced at the end, we have $\{s \mid \varnothing; s \in S\} = \bigsqcap_{t \in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$.*

Proof. Termination: Associate to each state $\{x_1 \textsf{ in } T_1, \dots x_n \textsf{ in } T_n\}; s$ the multiset $\{d_1, \dots, d_n\}$, where d_i is the maximum depth of terms occurring in T_i. $d_i = 0$ if $T_i = \varnothing$. Compare these multisets by the Dershowitz-Manna ordering [5]. Each rule strictly reduces them, which implies termination.

By the definitions of $\mathbf{rpc}_{\mathcal{R},\lambda}$ and $\sqcap$, $h(s_1, \dots, s_n) \in \bigsqcap_{t \in \{t_1,\dots,t_m\}} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$ iff $h \sim^{\rho_k}_{\mathcal{R},\gamma_k} \textsf{head}(t_k)$ with $\gamma_k \geqslant \lambda$ for all $1 \leqslant k \leqslant m$ and $s_i \in \bigsqcap_{t \in T_i} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$, where $T_i = \{t_k|_j \mid (i,j) \in \rho_k, 1 \leqslant k \leqslant m\}$, $1 \leqslant i \leqslant n$. Therefore, in the Rem rule, the instance of x (which is $h(y_1, \dots, y_n)$) is in $\bigsqcap_{t \in \{t_1,\dots,t_m\}} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$ iff for each $1 \leqslant i \leqslant n$ we can find an instance of y_i in $\bigsqcap_{t \in T_i} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$. If T_i is empty, it means that the i's argument of h is irrelevant for terms in $\{t_1, \dots, t_m\}$ and can be replaced by _. (Rem does it in a subsequent step.) Hence, in each success branch of the derivation tree, the algorithm $\mathfrak{C}$ computes one element of $\bigsqcap_{t \in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$. Branching at Red helps produce all elements of $\bigsqcap_{t \in T} \mathbf{rpc}_{\mathcal{R},\lambda}(t)$. □

It is easy to see how to use $\mathfrak{C}$ to decide the $(\mathcal{R}, \lambda)$-consistency of T: it is enough to find one successful branch in the $\mathfrak{C}$-derivation tree for $\{x \mathsf{~in~} T\}; x$. If there is no such branch, then T is not $(\mathcal{R}, \lambda)$-consistent. In fact, during the derivation we can even ignore the second component of the states.

4 Solving Generalization Problems

Now we can reformulate the anti-unification problem that will be solved in the remaining part of the paper. $\mathcal{R}$ is a proximity relation and λ is a cut value.

Given: $\mathcal{R}$, λ, and the ground terms $t_1, \ldots, t_n$, $n \geqslant 2$.

Find: a set S of tuples $(r, \sigma_1, \ldots, \sigma_n, \alpha_1, \ldots, \alpha_n)$ such that

- $\{r \mid (r, \ldots) \in \mathsf{S}\}$ is an $(\mathcal{R}, \lambda)$-mcsrg of $t_1, \ldots, t_n$,
- $r\sigma_i \simeq_{\mathcal{R},\lambda} t_i$ and $\alpha_i = \mathsf{gdub}_{\mathcal{R},\lambda}(r, t_i)$, $1 \leqslant i \leqslant n$, for each $(r, \sigma_1, \ldots, \sigma_n, \alpha_1, \ldots, \alpha_n) \in \mathsf{S}$.

(When $n = 1$, this is a problem of computing a relevant proximity class of a term.) Below we give a set of rules, from which one can obtain algorithms to solve the anti-unification problem for four versions of argument relations:

1. The most general (unrestricted) case; see algorithm $\mathfrak{A}_1$ below, the computed set of generalizations is an mcsrg;
2. Correspondence relations: using the same algorithm $\mathfrak{A}_1$, the computed set of generalizations is an mcsg;
3. Mappings: using a dedicated algorithm $\mathfrak{A}_2$, the computed set of generalizations is an mcsrg;
4. Correspondence mappings (bijections): using the same algorithm $\mathfrak{A}_2$, the computed set of generalizations is an mcsg.

Each of them has also the corresponding linear variant, computing minimal complete sets of (relevant) linear $(\mathcal{R}, \lambda)$-generalizations. They are denoted by adding the superscript lin to the corresponding algorithm name: $\mathfrak{A}_1^{\mathsf{lin}}$ and $\mathfrak{A}_2^{\mathsf{lin}}$.

For simplicity, we formulate the algorithms for the case $n = 2$. They can be extended for arbitrary n straightforwardly.

The main data structure in these algorithms is an anti-unification triple (AUT) $x : T_1 \triangleq T_2$, where T_1 and T_2 are finite *consistent* sets of ground terms. The idea is that x is a common generalization of all terms in $T_1 \cup T_2$. A configuration is a tuple $A; S; r; \alpha_1; \alpha_2$, where A is a set of AUTs to be solved, S is a set of solved AUTs (the store), r is the generalization computed so far, and the α's are the current approximations of generalization degree upper bounds of r for the input terms.

Before formulating the rules, we discuss one peculiarity of approximate generalizations:

Example 5. For a given $\mathcal{R}$ and λ, assume $\mathcal{R}(a, b) \geqslant \lambda$, $\mathcal{R}(b, c) \geqslant \lambda$, $h \sim_{\mathcal{R},\alpha}^{\{(1,1),(1,2)\}} f$ and $h \sim_{\mathcal{R},\beta}^{\{(1,1)\}} g$, where f is binary, g, h are unary, $\alpha \geqslant \lambda$ and $\beta \geqslant \lambda$. Then

- $h(b)$ is an $(\mathcal{R},\lambda)$-generalization of $f(a,c)$ and $g(a)$.
- x is the only $(\mathcal{R},\lambda)$-generalization of $f(a,d)$ and $g(a)$. One may be tempted to have h as the head of the generalization, e.g., $h(x)$, but x cannot be instantiated by any term that would be $(\mathcal{R},\lambda)$-close to both a and d, since in the given $\mathcal{R}$, d is $(\mathcal{R},\lambda)$-close only to itself. Hence, there would be no instance of $h(x)$ that is $(\mathcal{R},\lambda)$-close to $f(a,d)$. Since there is no other alternative (except h) for the common neighbor of f and g, the generalization should be a fresh variable x.

This example shows that generalization algorithms should take into account not only the heads of the terms to be generalized, but also should look deeper, to make sure that the arguments grouped together by the given argument relation have a common neighbor. This justifies the requirement of consistency of a set of arguments, the notion introduced in the previous section and used in the decomposition rule below.

4.1 Anti-unification for Unrestricted Argument Relations

Algorithms $\mathfrak{A}_1^{\mathsf{lin}}$ and $\mathfrak{A}_1$ use the rules below to transform configurations into configurations. Given $\mathcal{R}$, λ, and the ground terms t_1 and t_2, we create the initial configuration $\{x : \{t_1\} \triangleq \{t_2\}\}; \emptyset; x; 1; 1$ and apply the rules as long as possible. Note that the rules preserve consistency of AUTs. The process generates a finite complete tree of derivations, whose terminal nodes have configurations with the first component empty. We will show how from these terminal configurations one collects the result as required in the anti-unification problem statement.

Tri: **Trivial**

$\{x : \emptyset \triangleq \emptyset\} \uplus A;\ S;\ r;\ \alpha_1;\alpha_2 \Longrightarrow A;\ S;\ r\{x \mapsto \_\};\ \alpha_1;\alpha_2.$

Dec: **Decomposition**

$$\{x : T_1 \triangleq T_2\} \uplus A; S; r; \alpha_1; \alpha_2 \Longrightarrow$$
$$\{y_i : Q_{i1} \triangleq Q_{i2} \mid 1 \leqslant i \leqslant n\} \cup A; S; r\{x \mapsto h(y_1,\dots,y_n)\}; \alpha_1 \wedge \beta_1; \alpha_2 \wedge \beta_2,$$

where $T_1 \cup T_2 \neq \emptyset$; h is n-ary with $n \geqslant 0$; $y_1,\dots,y_n$ are fresh; and for $j = 1,2$, if $T_j = \{t^j_1,\dots,t^j_{m_j}\}$, then

- $h \sim^{\rho^j_k}_{\mathcal{R},\gamma^j_k} \mathsf{head}(t^j_k)$ with $\gamma^j_k \geqslant \lambda$ for all $1 \leqslant k \leqslant m_j$ and $\beta_j = \gamma^j_1 \wedge \cdots \wedge \gamma^j_{m_j}$ (note that $\beta_j = 1$ if $m_j = 0$),
- for all $1 \leqslant i \leqslant n$, $Q_{ij} = \cup_{k=1}^{m_j}\{t^j_k|_q \mid (i,q) \in \rho^j_k\}$ and is $(\mathcal{R},\lambda)$-consistent.

Sol: **Solving**

$\{x : T_1 \triangleq T_2\} \uplus A;\ S;\ r;\ \alpha_1;\alpha_2 \Longrightarrow A;\ \{x : T_1 \triangleq T_2\} \cup S;\ r;\ \alpha_1;\alpha_2,$
if Tri and Dec rules are not applicable. (It means that at least one $T_i \neq \emptyset$ and either there is no h as it is required in the Dec rule, or at least one Q_{ij} from Dec is not $(\mathcal{R},\lambda)$-consistent.)

Let expand be an *expansion operation* defined for sets of AUTs as

$$\mathsf{expand}(S) := \{x : \prod_{t \in T_1} \mathbf{rpc}_{\mathcal{R},\lambda}(t) \triangleq \prod_{t \in T_2} \mathbf{rpc}_{\mathcal{R},\lambda}(t) \mid x : T_1 \triangleq T_2 \in S\}.$$

Exhaustive application of the three rules above leads to configurations of the form $\emptyset; S; r; \alpha_1; \alpha_2$, where r is a linear term. These configurations are further postprocessed, replacing S by $\mathsf{expand}(S)$. We will use the letter E for expanded stores. Hence, terminal configurations obtained after the exhaustive rule application and expansion have the form $\emptyset; E; r; \alpha_1; \alpha_2$, where r is a linear term.[1] This is what Algorithm $\mathfrak{A}_1^{\mathsf{lin}}$ stops with.

To an expanded store $E = \{y_1 : Q_{11} \triangleq Q_{12}, \dots, y_n : Q_{n1} \triangleq Q_{n2}\}$ we associate two sets of substitutions $\Sigma_L(E)$ and $\Sigma_R(E)$, defined as follows: $\sigma \in \Sigma_L(E)$ (resp. $\sigma \in \Sigma_R(E)$) iff $\mathsf{dom}(\sigma) = \{y_1, \dots, y_n\}$ and $y_i\sigma \in Q_{i1}$ (resp. $y_i\sigma \in Q_{i2}$) for each $1 \leqslant i \leqslant n$. We call them the sets of *witness substitutions*.

Configurations containing expanded stores are called *expanded configurations*. From each expanded configuration $C = \emptyset; E; r; \alpha_1; \alpha_2$, we construct the set $\mathsf{S}(C) := \{(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \mid \sigma_1 \in \Sigma_L(E), \sigma_2 \in \Sigma_R(E)\}$.

Given an anti-unification problem $\mathcal{R}$, λ, t_1 and t_2, the *answer computed by Algorithm* $\mathfrak{A}_1^{\mathsf{lin}}$ is the set $\mathsf{S} := \cup_{i=1}^m \mathsf{S}(C_i)$, where $C_1, \dots, C_m$ are all of the final expanded configurations reached by $\mathfrak{A}_1^{\mathsf{lin}}$ for $\mathcal{R}$, λ, t_1, and t_2.[2]

Example 6. Assume a, b, c and d are constants with $b \sim_{\mathcal{R},0.5}^{\emptyset} c$, $c \sim_{\mathcal{R},0.6}^{\emptyset} d$, and f, g and h are respectively binary, ternary and quaternary function symbols with $h \sim_{\mathcal{R},0.7}^{\{(1,1),(3,2),(4,2)\}} f$ and $h \sim_{\mathcal{R},0.8}^{\{(1,1),(3,3)\}} g$. For the proximity relation $\mathcal{R}$ given in this way and $\lambda = 0.5$, Algorithm $\mathfrak{A}_1^{\mathsf{lin}}$ performs the following steps to anti-unify $f(a, b)$ and $g(a, c, d)$:

$$\begin{aligned}
&\{x : \{f(a,b)\} \triangleq \{g(a,c,d)\}\}; \emptyset; x; 1; 1 \Longrightarrow_{\mathsf{Dec}}\\
&\{x_1 : \{a\} \triangleq \{a\},\ x_2 : \emptyset \triangleq \emptyset,\ x_3 : \{b\} \triangleq \{d\},\\
&\qquad x_4 : \{b\} \triangleq \emptyset\}; \emptyset; h(x_1, x_2, x_3, x_4); 0.7; 0.8 \Longrightarrow_{\mathsf{Dec}}\\
&\{x_2 : \emptyset \triangleq \emptyset,\ x_3 : \{b\} \triangleq \{d\},\ x_4 : \{b\} \triangleq \emptyset\}; \emptyset; h(a, x_2, x_3, x_4); 0.7; 0.8 \Longrightarrow_{\mathsf{Tri}}\\
&\{x_3 : \{b\} \triangleq \{d\},\ x_4 : \{b\} \triangleq \emptyset\}; \emptyset; h(a, \_, x_3, x_4); 0.7; 0.8 \Longrightarrow_{\mathsf{Dec}}\\
&\{x_4 : \{b\} \triangleq \emptyset\}; \emptyset; h(a, \_, c, x_4); 0.5; 0.6.
\end{aligned}$$

Here Dec applies in two different ways, with the substitutions $\{x_4 \mapsto b\}$ and $\{x_4 \mapsto c\}$, leading to two final configurations: $\emptyset; \emptyset; h(a, \_, c, b); 0.5; 0.6$ and $\emptyset; \emptyset; h(a, \_, c, c); 0.5; 0.6$. The witness substitutions are the identity substitutions. We have $\mathcal{R}(h(a, \_, c, b), f(a, b)) = 0.5$, $\mathcal{R}(h(a, \_, c, b), g(a, c, d)) = 0.6$, $\mathcal{R}(h(a, \_, c, c), f(a, b)) = 0.5$, and $\mathcal{R}(h(a, \_, c, c), g(a, c, d)) = 0.6$.

If we had $h \sim_{\mathcal{R},0.7}^{\{(1,1),(1,2),(4,2)\}} f$, then the algorithm would perform only the Sol step, because in the attempt to apply Dec to the initial configuration, the set

[1] Note that no side of the AUTs in E in those configurations is empty due to the condition at the **Decomposition** rule requiring the Q_{ij}'s to be $(\mathcal{R}, \lambda)$-consistent.

[2] If we are interested only in linear generalizations *without witness substitutions*, there is no need in computing expanded configurations in $\mathfrak{A}_1^{\mathsf{lin}}$.

$Q_{11} = \{a, b\}$ is inconsistent: $\mathbf{rpc}_{\mathcal{R},\lambda}(a) = \{a\}$, $\mathbf{rpc}_{\mathcal{R},\lambda}(b) = \{b, c\}$, and, hence, $\mathbf{rpc}_{\mathcal{R},\lambda}(a) \sqcap \mathbf{rpc}_{\mathcal{R},\lambda}(b) = \varnothing$.

Algorithm $\mathfrak{A}_1$ is obtained by further transforming the expanded configurations produced by $\mathfrak{A}_1^{\mathsf{lin}}$. This transformation is performed by applying the **Merge** rule below as long as possible. Intuitively, its purpose is to make the linear generalization obtained by $\mathfrak{A}_1^{\mathsf{lin}}$ less general by merging some variables.

Mer: **Merge**

$$\varnothing; \{x_1 : R_{11} \triangleq R_{12},\, x_2 : R_{21} \triangleq R_{22}\} \uplus E;\, r;\, \alpha_1;\, \alpha_2 \Longrightarrow$$
$$\varnothing;\, \{y : Q_1 \triangleq Q_2\} \cup E;\, r\sigma;\, \alpha_1;\, \alpha_2,$$

where $Q_i = (R_{1i} \sqcap R_{2i}) \neq \varnothing$, $i = 1, 2$, y is fresh, and $\sigma = \{x_1 \mapsto y,\, x_2 \mapsto y\}$.

The answer computed by $\mathfrak{A}_1$ is defined similarly to the answer computed by $\mathfrak{A}_1^{\mathsf{lin}}$.

Example 7. Assume a, b are constants, f_1, f_2, g_1, and g_2 are unary function symbols, p is a binary function symbol, and h_1 and h_2 are ternary function symbols. Let λ be a cut value and $\mathcal{R}$ be defined as $f_i \sim_{\mathcal{R},\alpha_i}^{\{(1,1)\}} h_i$ and $g_i \sim_{\mathcal{R},\beta_i}^{\{(1,2)\}} h_i$ with $\alpha_i \geqslant \lambda$, $\beta_i \geqslant \lambda$, $i = 1, 2$. To generalize $p(f_1(a), g_1(b))$ and $p(f_2(a), g_2(b))$, we use $\mathfrak{A}_1$. The derivation starts as

$$\{x : \{p(f_1(a), g_1(b))\} \triangleq \{p(f_2(a), g_2(b))\}\};\, \varnothing;\, x;\, 1;\, 1 \Longrightarrow_{\mathsf{Dec}}$$
$$\{y_1 : \{f_1(a)\} \triangleq \{f_2(a)\},\, y_2 : \{g_1(b)\} \triangleq \{g_2(b)\}\};\, \varnothing;\, p(y_1, y_2);\, 1;\, 1 \Longrightarrow_{\mathsf{Sol}}^{2}$$
$$\varnothing;\, \{y_1 : \{f_1(a)\} \triangleq \{f_2(a)\},\, y_2 : \{g_1(b)\} \triangleq \{g_2(b)\}\};\, p(y_1, y_2);\, 1;\, 1.$$

At this stage, we expand the store, obtaining

$$\varnothing; \{y_1 : \{f_1(a), h_1(a, \_, \_)\} \triangleq \{f_2(a), h_2(a, \_, \_)\},$$
$$y_2 : \{g_1(b), h_1(\_, b, \_)\} \triangleq \{g_2(b), h_2(\_, b, \_)\}\};\, p(y_1, y_2);\, 1;\, 1.$$

If we had the standard intersection $\cap$ in the Mer rule, we would not be able to merge y_1 and y_2, because the obtained sets in the corresponding AUTs are disjoint. However, Mer uses $\sqcap$: we have $\{f_i(a), h_i(a, \_, \_)\} \sqcap \{g_i(b), h_i(\_, b, \_)\} = \{h_i(a, b, \_)\}$, $i = 1, 2$ and, therefore, can make the step

$$\varnothing; \{y_1 : \{f_1(a), h_1(a, \_, \_)\} \triangleq \{f_2(a), h_2(a, \_, \_)\},$$
$$y_2 : \{g_1(b), h_1(\_, b, \_)\} \triangleq \{g_2(b), h_2(\_, b, \_)\}\};\, p(y_1, y_2);\, 1;\, 1 \Longrightarrow_{\mathsf{Mer}}$$
$$\varnothing;\, \{z : \{h_1(a, b, \_)\} \triangleq \{h_2(a, b, \_)\}\};\, p(z, z);\, 1;\, 1.$$

Indeed, if we take the witness substitutions $\sigma_i = \{z \mapsto h_i(a, b, \_)\}$, $i = 1, 2$, and apply them to the obtained generalization, we get

$$p(z, z)\sigma_1 = p(h_1(a, b, \_), h_1(a, b, \_)) \simeq_{\mathcal{R},\lambda} p(f_1(a), g_1(b)),$$
$$p(z, z)\sigma_2 = p(h_2(a, b, \_), h_2(a, b, \_)) \simeq_{\mathcal{R},\lambda} p(f_2(a), g_2(b)).$$

Theorem 3. *Given $\mathcal{R}$, λ, and the ground terms t_1 and t_2, Algorithm $\mathfrak{A}_1$ terminates for $\{x : \{t_1\} \triangleq \{t_2\}\}; \varnothing; x; 1; 1$ and computes an answer set S such that*

1. *the set* $\{r \mid (r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}\}$ *is an* $(\mathcal{R}, \lambda)$*-mcsrg of* t_1 *and* t_2*,*
2. *for each* $(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}$ *we have* $\mathcal{R}(r\sigma_i, t_i) \leqslant \alpha_i = \mathsf{gdub}_{\mathcal{R},\lambda}(r, t_i)$*,* $i = 1, 2$*.*

Proof. Termination: Define the depth of an AUT $x : \{t_1, \ldots, t_m\} \triangleq \{s_1, \ldots, s_n\}$ as the depth of the term $f(g(t_1, \ldots, t_m), h(s_1, \ldots, s_n))$. The rules Tri, Dec, and Sol strictly reduce the multiset of depths of AUTs in the first component of the configurations. Mer strictly reduces the number of distinct variables in generalizations. Hence, these rules cannot be applied infinitely often and $\mathfrak{A}_1$ terminates.

In order to prove (1), we need to verify three properties:

- Soundness: If $(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}$, then r is a relevant $(\mathcal{R}, \lambda)$-generalization of t_1 and t_2.
- Completeness: If r' is a relevant $(\mathcal{R}, \lambda)$-generalization of t_1 and t_2, then there exists $(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}$ such that $r' \lesssim r$.
- Minimality: If r and r' belong to two tuples from S such that $r \neq r'$, then neither $r \prec_{\mathcal{R},\lambda} r'$ nor $r' \prec_{\mathcal{R},\lambda} r$.

Soundness: We show that each rule transforms an $(\mathcal{R}, \lambda)$-generalization into an $(\mathcal{R}, \lambda)$-generalization. Since we start from a most general $(\mathcal{R}, \lambda)$-generalization of t_1 and t_2 (a fresh variable x), at the end of the algorithm we will get an $(\mathcal{R}, \lambda)$-generalization of t_1 and t_2. We also show that in this process all irrelevant positions are abstracted by anonymous variables, to guarantee that each computed generalization is relevant.

Dec: The computed h is $(\mathcal{R}, \lambda)$-close to the head of each term in $T_1 \cup T_2$. Q_{ij}'s correspond to argument relations between h and those heads, and each Q_{ij} is $(\mathcal{R}, \lambda)$-consistent, i.e., there exists a term that is $(\mathcal{R}, \lambda)$-close to each term in Q_{ij}. It implies that $x\sigma = h(y_1, \ldots, y_n)$ $(\mathcal{R}, \lambda)$-generalizes all the terms from $T_1 \cup T_2$. Note that at this stage, $h(y_1, \ldots, y_n)$ might not yet be a relevant $(\mathcal{R}, \lambda)$-generalization of T_1 and T_2: if there exists an irrelevant position $1 \leqslant i \leqslant n$ for the $(\mathcal{R}, \lambda)$-generalization of T_1 and T_2, then in the new configuration we will have an AUT $y_i : \varnothing \triangleq \varnothing$.

Tri: When Dec generates $y : \varnothing \triangleq \varnothing$, the Tri rule replaces y by _ in the computed generalization, making it relevant.

Sol does not change generalizations.

Mer merges AUTs whose terms have *nonempty* intersection of **rpc**'s. Hence, we can reuse the same variable in the corresponding positions in generalizations, i.e., Mer transforms a generalization computed so far into a less general one.

Completeness: We prove a slightly more general statement. Given two finite consistent sets of ground terms T_1 and T_2, if r' is a relevant $(\mathcal{R}, \lambda)$-generalization for all $t_1 \in T_1$ and $t_2 \in T_2$, then starting from $\{x : T_1 \triangleq T_2\}; \varnothing; x; 1; 1$, Algorithm $\mathfrak{A}_1$ computes a $(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2)$ such that $r' \lesssim r$.

We may assume w.l.o.g. that r' is a relevant $(\mathcal{R}, \lambda)$-lgg. Due to the transitivity of $\lesssim$, completeness for such an r' will imply it for all terms more general than r'.

We proceed by structural induction on r'. If r' is a (named or anonymous) variable, the statement holds. Assume $r' = h(r'_1, \dots, r'_n)$, $T_1 = \{u_1, \dots, u_m\}$, and $T_2 = \{w_1, \dots, w_l\}$. Then h is such that $h \sim^{\rho_i}_{\mathcal{R},\beta_i} \mathsf{head}(u_i)$ for all $1 \leqslant i \leqslant m$ and $h \sim^{\mu_j}_{\mathcal{R},\gamma_j} \mathsf{head}(w_j)$ for all $1 \leqslant j \leqslant l$. Moreover, each r'_k is a relevant $(\mathcal{R}, \lambda)$-generalization of $Q_{k1} = \cup_{i=1}^{m} \{u_i|_q \mid (k, q) \in \rho_i\}$ and $Q_{k2} = \cup_{j=1}^{l} \{w_j|_q \mid (k, q) \in \mu_j\}$ and, hence, Q_{k1} and Q_{k2} are $(\mathcal{R}, \lambda)$-consistent. Therefore, we can perform a step by Dec, choosing $h(y_1, \dots, y_k)$ as the generalization term and $y_i : Q_{i1} \triangleq Q_{i2}$ as the new AUTs. By the induction hypothesis, for each $1 \leqslant i \leqslant n$ we can compute a relevant $(\mathcal{R}, \lambda)$-generalization r_i for Q_{i_1} and Q_{i2} such that $r'_i \lesssim r_i$.

If r' is linear, then the combination of the current Dec step with the derivations that lead to those r_i's computes a tuple $(r, \dots) \in \mathsf{S}$, where $r = h(r_1, \dots, r_n)$ and, hence, $r' \lesssim r$.

If r' is non-linear, assume without loss of generality that all occurrences of a shared variable z appear as the direct arguments of h: $z = r'_{k_1} = \dots = r'_{k_p}$ for $1 \leqslant k_1 < \dots < k_p \leqslant n$. Since r' is an lgg, Q_{k_i1} and Q_{k_i2} cannot be generalized by a non-variable term, thus, Tri and Dec are not applicable. Therefore, the AUTs $y_i : Q_{k_i1} \triangleq Q_{k_i2}$ would be transformed by Sol. Since all pairs Q_{k_i1} and Q_{k_i2}, $1 \leqslant i \leqslant p$, are generalized by the same variable, we have $\sqcap_{t \in Q_j} \mathbf{rpc}_{\mathcal{R},\lambda}(t) \neq \emptyset$, where $Q_j = \cup_{i=1}^{p} Q_{k_ij}$, $j = 1, 2$. Additionally, $r'_{k_1}, \dots, r'_{k_p}$ are all occurrences of z in r'. Hence, the condition of Mer is satisfied and we can extend our derivation with $p - 1$-fold application of this rule, obtaining $r = h(r_1, \dots, r_n)$ with $z = r_{k_1} = \dots = r_{k_p}$, implying $r' \lesssim r$.

Minimality: Alternative generalizations are obtained by branching in Dec or Mer. If the current generalization r is transformed by Dec into two generalizations r_1 and r_2 on two branches, then $r_1 = h_1(y_1, \dots, y_m)$ and $r_2 = h_2(z_1, \dots, z_n)$ for some h's, and fresh y's and z's. It may happen that $r_1 \lesssim_{\mathcal{R},\lambda} r_2$ or vice versa (if h_1 and h_2 are $(\mathcal{R}, \lambda)$-close to each other), but neither $r_1 \prec_{\mathcal{R},\lambda} r_2$ nor $r_2 \prec_{\mathcal{R},\lambda} r_1$ holds. Hence, the set of generalizations computed before applying Mer is minimal. Mer groups AUTs together maximally, and different groupings are not comparable. Therefore, variables in generalizations are merged so that distinct generalizations are not $\prec_{\mathcal{R},\lambda}$-comparable. Hence, (1) is proven.

As for (2), for $i = 1, 2$, from the construction in Dec follows $\mathcal{R}(r\sigma_i, t_i) \leqslant \alpha_i$. Mer does not change α_i, thus, $\alpha_i = \mathsf{gdub}_{\mathcal{R},\lambda}(r, t_i)$ also holds, since the way how α_i is computed corresponds exactly to the computation of $\mathsf{gdub}_{\mathcal{R},\lambda}(r, t_i)$: $r \lesssim_{\mathcal{R},\lambda} t_i$ and only the decomposition changes the degree during the computation. □

The corollary below is proved similarly to Theorem 3:

Corollary 1. *Given $\mathcal{R}$, λ, and the ground terms t_1 and t_2, Algorithm $\mathfrak{A}_1^{\mathsf{lin}}$ terminates for $\{x : \{t_1\} \triangleq \{t_2\}\}; \emptyset; x; 1; 1$ and computes an answer set* S *such that*

1. *the set $\{r \mid (r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}\}$ is a minimal complete set of relevant* linear *$(\mathcal{R}, \lambda)$-generalizations of t_1 and t_2,*
2. *for each $(r, \sigma_1, \sigma_2, \alpha_1, \alpha_2) \in \mathsf{S}$ we have $\mathcal{R}(r\sigma_i, t_i) \leqslant \alpha_i = \mathsf{gdub}_{\mathcal{R},\lambda}(r, t_i)$, $i = 1, 2$.*

4.2 Anti-unification with Correspondence Argument Relations

Correspondence relations make sure that for a pair of proximal symbols, no argument is irrelevant for proximity. Left- and right-totality of those relations guarantee that each argument of a term is close to at least one argument of its proximal term and the inverse relation remains a correspondence relation. Consequently, in the Dec rule of $\mathfrak{A}_1$, the sets Q_{ij} never get empty. Therefore, the Tri rule becomes obsolete and no anonymous variable appears in generalizations. As a result, the $(\mathcal{R},\lambda)$-mcsrg and the $(\mathcal{R},\lambda)$-mcsg coincide, and the algorithm computes a solution from which we get an $(\mathcal{R},\lambda)$-mcsg for the given anti-unification problem. The linear version $\mathfrak{A}_1^{\mathsf{lin}}$ works analogously.

4.3 Anti-unification with Argument Mappings

When the argument relations are mappings, we are able to design a more constructive method for computing generalizations and their degree bounds (Recall that our mappings are partial injective functions, which guarantees that their inverses are also mappings.) We denote this algorithm by $\mathfrak{A}_2$. The configurations stay the same as in before, but the AUTs in A will contain only empty or singleton sets of terms. In the store, we may still get (after the expansion) AUTs with term sets containing more than one element. Only the Dec rule differs from its previous counterpart, having a simpler condition:

Dec: **Decomposition**

$$\{x : T_1 \triangleq T_2\} \uplus A; S; r; \alpha_1; \alpha_2 \Longrightarrow$$
$$\{y_i : Q_{i1} \triangleq Q_{i2} \mid 1 \leqslant i \leqslant n\} \cup A; S; r\{x \mapsto h(y_1,\dots,y_n)\}; \alpha_1 \wedge \beta_1; \alpha_2 \wedge \beta_2,$$

where $T_1 \cup T_2 \neq \varnothing$; h is n-ary with $n \geqslant 0$; $y_1,\dots,y_n$ are fresh; for $j = 1,2$ and for all $1 \leqslant i \leqslant n$, if $T_j = \{t_j\}$ then $h \sim^{\pi_j}_{\mathcal{R},\beta_j} \mathsf{head}(t_j)$ and $Q_{ij} = \{t_j|_{\pi_j(i)}\}$, and if $T_j = \varnothing$ then $\beta_j = 1$ and $Q_{ij} = \varnothing$.

This Dec rule is equivalent to the special case of Dec for argument relations where $m_j \leqslant 1$. The new Q_{ij}'s contain at most one element (due to mappings) and, thus, are always $(\mathcal{R},\lambda)$-consistent. Various choices of h in Dec and alternatives in grouping AUTs in Mer cause branching in the same way as in $\mathfrak{A}_1$. It is easy to see that the counterparts of Theorem 3 hold for $\mathfrak{A}_2$ and $\mathfrak{A}_2^{\mathsf{lin}}$ as well.

A special case of this fragment of anti-unification is anti-unification for similarity relations in fully fuzzy signatures from [1]. Similarity relations are min-transitive proximity relations. The position mappings in [1] can be modeled by our argument mappings, requiring them to be total for symbols of the smaller arity and to satisfy the similarity-specific consistency restrictions from [1].

4.4 Anti-unification with Correspondence Argument Mappings

Correspondence argument mappings are bijections between arguments of function symbols of the same arity. For such mappings, if $h \simeq^{\pi}_{\mathcal{R},\lambda} f$ and h is n-ary, then f is also n-ary and π is a permutation of $(1,\dots,n)$. Hence, $\mathfrak{A}_2$ combines in this case the properties of $\mathfrak{A}_1$ for correspondence relations (Sect. 4.2) and of $\mathfrak{A}_2$ for argument mappings (Sect. 4.3): all generalizations are relevant, computed answer gives an mcsg of the input terms, and the algorithm works with term sets of cardinality at most 1.

5 Remarks About the Complexity

The proximity relation $\mathcal{R}$ can be naturally represented as an undirected graph, where the vertices are function symbols and an edge between them indicates that they are proximal. Graphs induced by proximity relations are usually sparse. Therefore we can represent them by (sorted) adjacency lists. In the adjacency lists, we can also accommodate the argument relations and proximity degrees.

In the rest of this section we use the following notation:

- n: the size of the input (number of symbols) of the corresponding algorithms,
- Δ: the maximum degree of $\mathcal{R}$ considered as a graph,
- $\mathfrak{a}$: the maximum arity of function symbols that occur in $\mathcal{R}$.
- $m^{\bullet n}$: a function defined on natural numbers m and n such that $1^{\bullet n} = n$ and $m^{\bullet n} = m^n$ for $m \neq 1$.

We assume that the given anti-unification problem is represented as a completely shared directed acyclic graph (dag). Each node of the dag has a pointer to the adjacency list (with respect to $\mathcal{R}$) of the symbol in the node.

Theorem 4. *Time complexities of $\mathfrak{C}$ and the linear versions of the generalization algorithms are as follows:*

- *$\mathfrak{C}$ for argument relations and $\mathfrak{A}_1^{\mathsf{lin}}$:* $O(n \cdot \varDelta \cdot \varDelta^{\bullet \mathfrak{a}^{\bullet n}})$,
- *$\mathfrak{C}$ for argument mappings and $\mathfrak{A}_2^{\mathsf{lin}}$:* $O(n \cdot \varDelta \cdot \varDelta^{\bullet n})$.

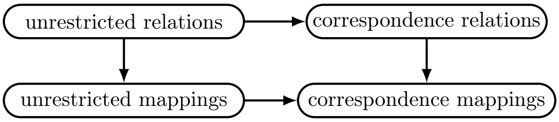
Proof (Sketch). In $\mathfrak{C}$, in the case of argument relations, an application of the Red rule to a state $\mathbf{I}; s$ replaces one element of $\mathbf{I}$ of size m by at most $\mathfrak{a}$ new elements, each of them of size $m-1$. Hence, one branch in the search tree for $\mathfrak{C}$, starting from a singleton set $\mathbf{I}$ of size n, will have the length at most $l = \sum_{i=0}^{n-1} \mathfrak{a}^i$. At each node on it there are at most $\varDelta$ choices of applying Red with different h's, which gives the total size of the search tree to be at most $\sum_{i=0}^{l-1} \varDelta^i$, i.e., the number of steps performed by $\mathfrak{C}$ in the worst case is $O(\varDelta^{\bullet \mathfrak{a}^{\bullet n}})$. Those different h's are obtained by intersecting the proximity classes of the heads of terms $\{t_1, \dots, t_m\}$ in the Red rule. In our graph representation of the proximity relation, proximity classes of symbols are exactly the adjacency lists of those symbols which we assume are sorted. Their maximal length is $\varDelta$. Hence, the work to be done at each node of the search tree of $\mathfrak{C}$ is to find the intersection of at most n sorted lists, each containing at most $\varDelta$ elements. It needs $O(n \cdot \varDelta)$ time. It gives the time complexity $O(n \cdot \varDelta \cdot \varDelta^{\bullet \mathfrak{a}^{\bullet n}})$ of $\mathfrak{C}$ for the relation case.

In the mapping case, an application of the Red rule to a state $\mathbf{I}; s$ replaces one element of $\mathbf{I}$ of size m by at most $\mathfrak{a}$ new elements of the *total* size $m-1$. Therefore, the maximal length of a branch is n, the branching factor is $\varDelta$, and the amount of work at each node, like above, is $O(n \cdot \mathit{\Delta})$. Hence, the number of steps in the worst case is $O(\mathit{\Delta}^{\bullet n})$ and the time complexity of $\mathfrak{C}$ is $O(n \cdot \mathit{\Delta} \cdot \mathit{\Delta}^{\bullet n})$.

The fact that consistency check is incorporated in the Dec rule in $\mathfrak{A}_1^{\mathsf{lin}}$ can be used to guide the application of this rule, using the values memoized by the previous applications of Red. The very first time, the appropriate h in Dec is chosen arbitrarily. In any subsequent application of this rule, h is chosen according to the result of the Red rule that has already been applied to the arguments of the current AUT for their consistency check, as required by the condition of Dec. In this way, the applications of Dec and Sol will correspond to the applications of Red. There is a natural correspondence between the applications of Rem and Tri rules. Therefore, $\mathfrak{A}_1^{\mathsf{lin}}$ will have the search tree analogous to that of $\mathfrak{C}$. Hence the complexity of $\mathfrak{A}_1^{\mathsf{lin}}$ is $O(n \cdot \mathit{\Delta} \cdot \mathit{\Delta}^{\bullet \mathfrak{a}^{\bullet n}})$. $\mathfrak{A}_2^{\mathsf{lin}}$ does not call the consistency check, but does the same work as $\mathfrak{C}$ and, hence, has the same complexity $O(n \cdot \mathit{\Delta} \cdot \mathit{\Delta}^{\bullet n})$. □

6 Discussion and Conclusion

The diagram below illustrates the connections between different anti-unification problems based on argument relations:



The arrows indicate the direction from more general problems to more specific ones. For the unrestricted cases (left column) we compute mcsrg's. For correspondence relations and correspondence mappings (right column), mcsg's are computed. (In fact, for them, the notions of mcsrg and mcsg coincide). The algorithms for relations (upper row) are more involved than those for mappings (lower row): Those for relations deal with AUTs containing arbitrary sets of terms, while for mappings, those sets have cardinality at most one, thus simplifying the conditions in the rules. Moreover, the two cases in the lower row generalize the existing anti-unification problems:

- the unrestricted mappings case generalizes the problem from [1] by extending similarity to proximity and relaxing the smaller-side-totality restriction;
- the correspondence mappings case generalizes the problem from [9] by allowing permutations between arguments of proximal function symbols.

All our algorithms can be easily turned into anti-unification algorithms for crisp tolerance relations[3] by taking lambda-cuts and ignoring the computation of the approximation degrees. Besides, they are modular and can be used to compute only linear generalizations by just skipping the merging rule. We provided complexity estimations for the algorithms that compute linear generalizations (that often are of practical interest).

[3] Tolerance: reflexive, symmetric, not necessarily transitive relation. According to Poincaré, a fundamental notion for mathematics applied to the physical world.

In this paper, we did not consider cases when the same pair of symbols is related to each other by more than one argument relation. Our results can be extended to them, that would open a way towards approximate anti-unification modulo background theories specified by shallow collapse-free axioms. Another interesting direction of future work would be extending our results to quantitative algebras [10] that also deal with quantitative extensions of equality.

Acknowledgments. Supported by the Austrian Science Fund, project P 35530.

References

1. Aït-Kaci, H., Pasi, G.: Fuzzy lattice operations on first-order terms over signatures with similar constructors: a constraint-based approach. Fuzzy Sets Syst. **391**, 1–46 (2020). https://doi.org/10.1016/j.fss.2019.03.019
2. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
3. Bader, J., Scott, A., Pradel, M., Chandra, S.: Getafix: learning to fix bugs automatically. Proc. ACM Program. Lang. **3**(OOPSLA), 159:1–159:27 (2019). https://doi.org/10.1145/3360585
4. Barwell, A.D., Brown, C., Hammond, K.: Finding parallel functional pearls: Automatic parallel recursion scheme detection in Haskell functions via anti-unification. Future Gener. Comput. Syst. **79**, 669–686 (2018). https://doi.org/10.1016/j.future.2017.07.024
5. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM **22**(8), 465–476 (1979). https://doi.org/10.1145/359138.359142
6. Galitsky, B.: Developing Enterprise Chatbots - Learning Linguistic Structures. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-04299-8
7. Kirbas, S., et al.: On the introduction of automatic program repair in Bloomberg. IEEE Softw. **38**(4), 43–51 (2021). https://doi.org/10.1109/MS.2021.3071086
8. Kutsia, T., Pau, C.: A framework for approximate generalization in quantitative theories. RISC Report Series 22-04, Research Institute for Symbolic Computation, Johannes Kepler University Linz (2022). https://doi.org/10.35011/risc.22-04
9. Kutsia, T., Pau, C.: Matching and generalization modulo proximity and tolerance relations. In: Özgün, A., Zinova, Y. (eds.) TbiLLC 2019. LNCS, vol. 13206, pp. 323–342. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-98479-3_16
10. Mardare, R., Panangaden, P., Plotkin, G.D.: Quantitative algebraic reasoning. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2016, pp. 700–709. ACM (2016). https://doi.org/10.1145/2933575.2934518
11. Mehta, S., et al.: Rex: preventing bugs and misconfiguration in large services using correlated change analysis. In: Bhagwan, R., Porter, G. (eds.) 17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, 25–27 February 2020, pp. 435–448. USENIX Association (2020). https://www.usenix.org/conference/nsdi20/presentation/mehta
12. Plotkin, G.D.: A note on inductive generalization. Mach. Intell. **5**(1), 153–163 (1970)
13. Raza, M., Gulwani, S., Milic-Frayling, N.: Programming by example using least general generalizations. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, 27–31 July 2014, Québec City, Québec, Canada, pp. 283–290. AAAI Press (2014)

14. Reynolds, J.C.: Transformational systems and the algebraic structure of atomic formulas. Mach. Intell. **5**(1), 135–151 (1970)
15. Rolim, R., Soares, G., Gheyi, R., D'Antoni, L.: Learning quick fixes from code repositories. CoRR abs/1803.03806 (2018). http://arxiv.org/abs/1803.03806

Guiding an Automated Theorem Prover with Neural Rewriting

Jelle Piepenbrock[1,2](✉), Tom Heskes[2], Mikoláš Janota[1], and Josef Urban[1]

[1] Czech Technical University in Prague, Prague, Czech Republic
Jelle.Piepenbrock@cvut.cz
[2] Radboud University, Nijmegen, The Netherlands

Abstract. Automated theorem provers (ATPs) are today used to attack open problems in several areas of mathematics. An ongoing project by Kinyon and Veroff uses Prover9 to search for the proof of the Abelian Inner Mapping (AIM) Conjecture, one of the top open conjectures in quasigroup theory. In this work, we improve Prover9 on a benchmark of AIM problems by neural synthesis of useful alternative formulations of the goal. In particular, we design the 3SIL (stratified shortest solution imitation learning) method. 3SIL trains a neural predictor through a reinforcement learning (RL) loop to propose correct rewrites of the conjecture that guide the search.

3SIL is first developed on a simpler, Robinson arithmetic rewriting task for which the reward structure is similar to theorem proving. There we show that 3SIL outperforms other RL methods. Next we train 3SIL on the AIM benchmark and show that the final trained network, deciding what actions to take within the equational rewriting environment, proves 70.2% of problems, outperforming Waldmeister (65.5%). When we combine the rewrites suggested by the network with Prover9, we prove 8.3% more theorems than Prover9 in the same time, bringing the performance of the combined system to 90%.

Keywords: Automated theorem proving · Machine learning

1 Introduction

Machine learning (ML) has recently proven its worth in a number of fields, ranging from computer vision [17], to speech recognition [15], to playing games [28,40] with *reinforcement learning* (RL) [45]. It is also increasingly applied in automated and interactive theorem proving. Learned predictors have been used for premise selection [1] in hammers [6], to improve clause selection in saturation-based theorem provers [9], to synthesize functions in higher-order logic [12], and to guide connection-tableau provers [21] and interactive theorem provers [2,5,14].

Future growth of the knowledge base of mathematics and the complexity of mathematical proofs will increase the need for proof checking and its better computer support and automation. Simultaneously, the growing complexity of software will increase the need for formal verification to prevent failure modes [10].

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 597–617, 2022.
https://doi.org/10.1007/978-3-031-10769-6_35

Automated theorem proving and mathematics will benefit from more advanced ML integration. One of the mathematical subfields that makes substantial use of automated theorem provers is the field of quasigroup and loop theory [32].

1.1 Contributions

In this paper, we propose to use a neural network to suggest lemmas to the Prover9 [25] ATP system by rewriting parts of the conjecture (Sect. 2). We test our method on a dataset of theorems collected in the work on the Abelian Inner Mapping (AIM) Conjecture [24] in loop theory. For this, we use the AIMLEAP proof system [7] as a reinforcement learning environment. This setup is described in Sect. 3. For development we used a simpler Robinson arithmetic rewriting task (Sect. 4). With the insights derived from this and a comparison with other methods, we describe our own 3SIL method in Sect. 5. We use a neural network to process the state of the proving attempt, for which the architecture is described in Sect. 6. The results on the Robinson arithmetic task are described in Sect. 7.1. We show our results on the AIMLEAP proving task, both using our predictor as a stand-alone prover and by suggesting lemmas to Prover9 in Sect. 7.2. Our contributions are:

1. We propose a training method for reinforcement learning in theorem proving settings: *stratified shortest solution imitation learning* (3SIL). This method is suited to the structure of theorem proving tasks. This method and the reasoning behind it is explained in Sect. 5.
2. We show that 3SIL outperforms other baseline RL methods on a simpler, Robinson arithmetic rewriting task for which the reward structure is similar to theorem proving (Sect. 7.1).
3. We show that a standalone neurally guided prover trained by the 3SIL method outperforms the hand-engineered Waldmeister prover on the AIMLEAP benchmark (Sect. 7.2).
4. We show that using a neural rewriting step that suggests rephrased versions of the conjecture to be added as lemmas improves the ATP performance on equational problems (Sects. 2 and 7.2).

2 ATP and Suggestion of Lemmas by Neural Rewriting

Saturation-based ATPs make use of the *given clause* [30] algorithm, which we briefly explain as background. A problem is expressed as a conjunction of many initial clauses (i.e., the clausified axioms and the negated goal which is always an equation in the AIM dataset). The algorithm starts with all the initial clauses in the *unprocessed set*. We then pick a clause from this set to be the given clause and move it to the *processed set* and do all inferences with the clauses in the processed set. The newly inferred clauses are added to the unprocessed set. This concludes one iteration of the algorithm, after which we pick a new given

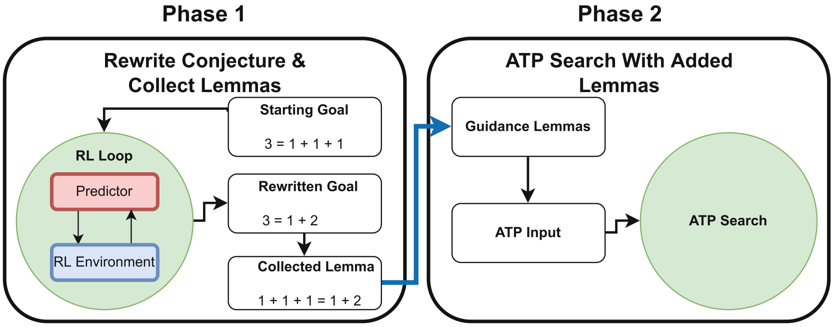


Fig. 1. Schematic representation of the proposed guidance method. In the first phase, we run a reinforcement learning loop to propose actions that rewrite a conjecture. This predictor is trained using the AIMLEAP proof environment. We collect the rewrites of the LHS and RHS of the conjecture. In the second phase, we add the rewrites to the ATP search input, to act as guidance. In this specific example, we only rewrote the conjecture for 1 step, but the added guidance lemmas are in reality the product of many steps in the RL loop.

clause and repeat [23]. Typically, this approach is designed to be *refutationally complete*, i.e., the algorithm is guaranteed to eventually find a contradiction if the original goal follows from the axioms.

This process can produce a lot of new clauses and the search space can become quite large. In this work, we modify the standard loop by adding useful lemmas to the initial clause set. These lemmas are proposed by a neural network that was trained *from zero knowledge* to rewrite the left- and right-hand sides of the initial goal to make them equal by using the axioms as the available rewrite actions. Even though the neural rewriting might not fully succeed, the rewrites produced by this process are likely to be useful as additional lemmas when added to the problem. This idea is schematically represented in Fig. 1.

3 AIM Conjecture and the AIMLEAP RL Environment

Automated theorem proving has been applied in the theory surrounding the Abelian Inner Mapping Conjecture, known as the AIM Conjecture. This is one of the top open conjectures in quasigroup theory. Work on the conjecture has been going on for more than a decade. Automated theorem provers use hundreds of thousands of inference steps when run on problems from this theory.

As a testbed for our machine learning and prover guidance methods we use a previously published dataset of problems generated by the AIM conjecture [7]. The dataset comes with a simple prover called AIMLEAP that can take machine learning advice.[1] We use this system as an RL environment. AIMLEAP keeps the state and carries out the cursor movements (the cursor determines the location of the rewrite) and rewrites that a neural predictor chooses.

[1] https://github.com/ai4reason/aimleap.

The AIM conjecture concerns specific structures in *loop theory* [24]. A loop is a quasigroup with an identity element. A quasigroup is a generalization of a group that does not preserve associativity. This manifests in the presence of two different 'division' operators, one left-division (\) and one right-division (/). We briefly explain the conjecture to show the nature of the data.

For loops, three *inner mapping functions* (left-translation L, right-translation R, and the mapping T) are:

$$L(u,x,y) := (y*x)\backslash(y*(x*u)) \qquad T(u,x) := x\backslash(u*x)$$
$$R(u,x,y) := ((u*x)*y)/(x*y)$$

These mappings can be seen as measures of the deviation from commutativity and associativity. The conjecture concerns the consequences of these three inner mapping functions forming an Abelian (commutative) group. There are two more notions, that of the *associator* function a and the *commutator* function K:

$$a(x,y,z) := (x*(y*z))\backslash((x*y)*z) \qquad K(x,y) := (y*x)/(x*y)$$

From these definitions, the conjecture can be stated. There are two parts to the conjecture. For both parts, the following equalities need to hold for all u, v, x, y, and z:

$$a(a(x,y,z),u,v) = 1 \qquad a(x,a(y,z,u),v) = 1 \qquad a(x,y,a(z,u,v)) = 1$$

where 1 is the identity element. These are necessary, but not sufficient for the two main parts of the conjecture. The first part of the conjecture asks whether a loop modulo its center is a group. In this context, the *center* is the set of all elements that commute with all other elements. This is the case if

$$K(a(x,y,z),u) = 1.$$

The second part of the conjecture asks whether a loop modulo its nucleus is an Abelian group. The *nucleus* is the set of elements that associate with all other elements. This is the case if

$$a(K(x,y),z,u) = 1 \qquad a(x,K(y,z),u) = 1 \qquad a(x,y,K(z,u)) = 1$$

3.1 The AIMLEAP RL Environment

Currently, work in this area is done using automated theorem provers such as *Prover9* [24,25]. This has led to some promising results, but the search space is enormous. The main strategy for proving the AIM conjecture thus far has been to prove weaker versions of the conjecture (using additional assumptions) and then import crucial proof steps into the stronger version of the proof. The *Prover9* theorem prover is especially suited to this approach because of its well-established *hints* mechanism [48]. The AIMLEAP dataset is derived from this

Prover9 approach and contains around 3468 theorems that can be proven with the supplied definitions and lemmas [7].

There are 177 possible actions in the AIMLEAP environment [7]. We handle the proof state as a tree, with the root node being an equality node. Three actions are cursor movements, where the cursor can be moved to an argument of the current position. The other actions all rewrite the current term at the cursor position with various axioms, definitions and lemmas that hold in the AIM context. As an example, this is one of the theorems in the dataset ($\backslash$ and $=$ are part of the language):

$$T(T(T(x,T(x,y)\backslash 1),T(x,y)\backslash 1),y) = T((T(x,y)\backslash 1)\backslash 1,T(x,y)\backslash 1)\,.$$

The task of the machine learning predictor is to process the proof state and recognize which actions are most likely to lead to a proof, meaning that the two sides of the starting equation are equal according to the AIMLEAP system. The only feedback that the environment gives is whether a proof has been found or not: there is no intermediate reward (i.e. rewards are *sparse*). The ramifications of this are further discussed in Sect. 5.1.

4 Rewriting in Robinson Arithmetic as an RL Task

To develop a machine learning method that can help solve equational theorem proving problems, we considered a simpler arithmetic task, which also has a tree-structured input and *a sparse reward structure*: the normalization of Robinson arithmetic expressions. The task is to normalize a mathematical expression to one specific form. This task has been implemented as a Python RL environment, which we make available.[2] The learning environment incorporates an existing dataset, constructed by Gauthier for RL experiments in the interactive theorem prover HOL4 [11]. Our RL setup for the task is also modeled after [11].

In more detail, the formalism that we use as an RL environment is Robinson arithmetic (RA). RA is a simple arithmetic theory. Its language contains the successor function S, addition $+$ and multiplication $*$ and one constant, the 0. The theory considers only non-negative numbers and we only use four axioms of RA. Numbers are represented by the constant 0 with the appropriate number of successor functions applied to it. The task for the agent is to rewrite an expression until there are only nodes of the successor or 0 types. Effectively, we are asking the agent to calculate the value of the expression. As an example, $S(S(0)) + S(0)$, representing $2 + 1$, needs to be rewritten to $S(S(S(0)))$.

The expressions are represented as a tree data structure. Within the environment, there are seven different rewrite actions available to the agent. The four axioms (equations) defining these actions are $x + 0 = x$, $x + S(y) = S(x + y)$, $x * 0 = 0$ and $x * S(y) = (x * y) + x$, where the agent can apply the equations in either direction. There is one exception: the multiplication by 0 cannot be applied from right to left, as this would require the agent to introduce a fresh

[2] https://github.com/learningeqtp/rewriteRL.

term which is out of scope for the current work. The place where the rewrite is applied is denoted by the location of the *cursor* in the expression tree.

In addition to the seven rewrite actions, the agent can move the cursor to one of the children of the current cursor node. This gives a total number of nine actions. Moving to a child of a node with only one child counts as moving to the left child. After a rewriting action, the cursor is reset to the root of the expression. More details on the actions are in the RewriteRL repository.

5 Reinforcement Learning Methods

This section describes the reinforcement learning methods, while Sect. 6 then further explains the particular neural architectures that are trained in the RL loops. We first briefly explain here the approaches that we used as reinforcement learning (RL) baselines, then we go into detail about the proposed 3SIL method.

5.1 Reinforcement Learning Baselines

General RL Setup. For comparison, we used implementations of four established reinforcement learning baseline methods. In reinforcement learning, we consider an *agent* that is acting within an *environment*. The agent can take actions a from the action-space $\mathcal{A}$ to change the state $s \in \mathcal{S}$ of the environment. The agent can be rewarded for certain actions taken in a certain states, with reward given by the *reward function* $\mathcal{R} : (\mathcal{S} \times \mathcal{A}) \rightarrow \mathbb{R}$. The behavior of the environment is given by the *state transition function* $\mathcal{P} : (\mathcal{S} \times \mathcal{A}) \rightarrow \mathcal{S}$. The history of the agent's actions and the environments states and rewards at each timestep t are collected in tuples (s_t, a_t, r_t). For a given history of a certain agent within an environment, we call the list of tuples (s_t, a_t, r_t) describing this history an *episode*. The *policy function* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ allows the agent to decide which action to take. The agent's goal is to maximize the return R: the sum of discounted rewards $\sum_{t \geq 0} \gamma^t r_t$, where γ is a *discount factor* that allows control over how heavily rewards further in the future should be weighted. We will use R_t when we mean R, but calculated only from rewards from timestep t on. In the end, we are thus looking for a policy function π that maximizes the sum R of (discounted) expected rewards [45].

In our setting, every proof attempt (in the AIM setting) or normalization attempt (in the Robinson arithmetic setting) corresponds to an episode. The reward structure of theorem proving is such that there is only a reward of 1 at the end of a successful episode (i.e. a proof was found in AIM). Unsuccessful episodes get a reward of 0 at every timestep t.

A2C. The first method, *Advantage Actor-Critic*, or *A2C* [27] contains ideas on which the other three RL baseline methods build, so we will go into more detail for this method, while keeping the explanation for the other methods brief. For details we refer to the corresponding papers.

A2C attempts to find suitable parameters for an agent by minimizing a *loss function* consisting of two parts:

$$\mathcal{L} = \mathcal{L}^{\text{A2C}}_{\text{policy}} + \mathcal{L}^{\text{A2C}}_{\text{value}} .$$

In addition to the policy function π, the agent has access to a *value function* $\mathcal{V} : \mathcal{S} \rightarrow \mathbb{R}$, that predicts the sum of future rewards obtained when given a state. In practice, both the policy and the value function are computed by a neural network *predictor*. The parameters of the predictor are set by *stochastic gradient descent* to minimize $\mathcal{L}$. The set of parameters of the predictor that defines the policy function π is named θ, while the parameters that define the value function are named μ. The first part of the loss is the *policy loss*, which for one time step has the form

$$\mathcal{L}^{\text{A2C}}_{\text{policy}} = -\log \pi_\theta(a_t|s_t) A(s_t, a_t) ,$$

where $A(s, a)$ is the *advantage function*. The advantage function can be formulated in multiple ways, but the simplest is as $R_t - \mathcal{V}_\mu(s_t)$. That is to say: the advantage of an action in a certain state is the difference between the discounted rewards R_t after taking that action and the value estimate of the current state.

Minimizing $\mathcal{L}^{\text{A2C}}_{\text{policy}}$ amounts to maximizing the log probability of predicting actions that are judged by the advantage function to lead to high reward.

The value estimates $V_\mu(s)$ for computing the advantage function are supplied by the *value predictor* V_μ with parameters μ, which is trained using the loss:

$$\mathcal{L}^{\text{A2C}}_{\text{value}} = \frac{1}{2} \left(R_t - \mathcal{V}_\mu(s_t) \right)^2 ,$$

which minimizes the advantage function. The logic of this is that the value estimate at timestep t, $\mathcal{V}_\mu(s_t)$, will learn to incorporate the later rewards R_t, ensuring that when later seeing the same state, the possible future reward will be considered. Note that the sets of parameters θ and μ are not necessarily disjoint (see Sect. 6).

Note how the above equations are affected if there is no non-zero reward r_t obtained at any timestep. In that case, the value function $\mathcal{V}_\mu(s_t)$ will estimate (correctly) that any state will get 0 reward, which means that the advantage function $A(s, a)$ will also be 0 everywhere. This means that $\mathcal{L}^{\text{A2C}}_{\text{policy}}$ will be 0 in most cases, which will lead to no or little change in the parameters of the predictor: learning will be very slow. This is the difficult aspect of the structure of theorem proving: there is only reward at the end of a successful proof, and nowhere else. This implies a possible strategy is to imitate successful episodes, without a value function. In this case, we would only need to train a *policy function*, and no approximate *value function*. This an aspect we explore in the design of our own method 3SIL, which we will explain shortly.

Compared to two-player games, such as chess and go, for which many approaches have been tailored and successfully used [41], theorem-proving has the property that it is hard to collect useful examples to learn from, as only successful proofs are likely to contain useful knowledge. In chess or go, however, one player almost always wins and the other loses, which means that we can at least learn from the difference between the two strategies used by those players. As an example, we executed 2 million random proof attempts on the AIMLEAP environment, which led to 300 proofs to learn from, whereas in a two-player setting like chess, we would get 2 million games in which one player would likely win.

ACER. The second RL baseline method we tested in our experiments is ACER, *Actor-Critic with Experience Replay* [49]. This approach can make use of data from older episodes to train the current predictor. ACER applies corrections to the value estimates so that data from old episodes may be used to train the current policy. It also uses trust region policy optimization [35] to limit the size of the policy updates. This method is included as a baseline to check if using a larger replay buffer to update the parameters would be advantageous.

PPO. Our third RL baseline is the widely used *proximal policy optimization* (PPO) algorithm [36]. It restricts the size of the parameter update to avoid causing a large difference between the original predictor's behavior and the updated version's behavior. The method is related to the above trust region policy optimization method. In this way, PPO addresses the training instability of many reinforcement learning approaches. It has been used in various settings, for example complex video games [4]. With its versatility, the PPO algorithm is well-positioned. We use the PPO algorithm with clipped objective, as in [36].

SIL-PAAC. Our final RL baseline uses only the transitions with positive advantage to train on for a portion of the training procedure, to learn more from good episodes. This was proposed as *self-imitation learning* (SIL) [29]. To avoid confusion with the method that we are proposing, we extend the acronym to SIL-PAAC, for positive advantage actor-critic. This algorithm outperformed A2C on the sparse-reward task Montezuma's Revenge (a puzzle game). As theorem proving has a sparse reward structure, we included SIL-PAAC as a baseline. More information about the implementations for the baselines can be found in the Implementation Details section at the end of this work.

5.2 Stratified Shortest Solution Imitation Learning

We introduce stratified shortest solution imitation learning (**3SIL**) to tackle the equational theorem proving domain. It learns to explicitly imitate the actions taken during the shortest solutions found for each problem in the dataset. We do this by minimizing the cross-entropy $-\log p(a_{solution}|s_t)$ between the predictor output and the actions taken in the shortest solution. This is in contrast to the baseline methods, where value functions are used to judge the utility of decisions.

In our procedure this is not the case. Instead, we build upon the assumption for data selection that shorter proofs are better in the context of theorem proving

Algorithm 1. CollectEpisode

```
**Input:** problem $p$, policy $\pi_\theta$, problem history H
Generate episode by following noisy version of $\pi_\theta$ on $p$
**If** solution, add list of tuples $(s, a)$ to H[$p$]
Keep $k$ shortest solutions in H[$p$]
```

Algorithm 2. 3SIL

```
**Input:** set of problems P, randomly initialized policy $\pi_\theta$, batch size $B$, number of batches NB, problem history H, number of warmup episodes $m$, number of episodes $f$, max epochs ME
**Output:** trained policy $\pi_\theta$, problem history H
**for** $e = 0$ **to** ME $- 1$ **do**
    **if** $e = 0$ **then** num $= m$ **else** num $= f$
    **for** $i = 0$ **to** num $- 1$ **do**
        CollectEpisode(sample(P), $\pi_\theta$, H) (Algorithm 1)
    **end for**
    **for** $i = 0$ **to** NB $- 1$ **do**
        Sample $B$ tuples $(s,\ a)$ with uniform probability for each problem from H
        Update $\theta$ to lower $-\sum_{b=0}^{B} \log \pi_\theta(a_b|s_b)$ by gradient descent
    **end for**
**end for**
```

and expression normalization. In a sense, we value decisions from shorter proofs more and explicitly imitate those transitions. We keep a history H for each problem, where we store the current shortest solution (states seen and actions taken) found for that problem in the training dataset. We can also store multiple shortest solutions for each problem if there are multiple strategies for a proof (the number of solutions kept is governed by the parameter k).

During training, in the case $k = 1$, we sample state-action pairs from each problem's current shortest solution at an equal probability (if a solution was found). To be precise, we first randomly pick a theorem for which we have a solution, and then randomly sample one transition from the shortest encountered solution. This directly counters one of the phenomena that we had observed: the training examples for the baseline methods tend to be dominated by very long episodes (as they contribute more states and actions). This *stratified* sampling method ensures that problems with short proofs get represented equally in the training process.

The 3SIL algorithm is described in more detail in Algorithm 2. Sampling from a noisy version of policy π_θ means that actions are sampled from the predictor-defined distribution and in 5% of cases a random valid action is selected. This is also known as the ϵ-greedy policy (with ϵ at 0.05).

Related Methods. Our approach is similar to the imitation learning algorithm DAGGER (Dataset Aggregation), which was used for several games [34] and modified for branch-and-bound algorithms in [16]. The behavioral cloning (BC) technique used in robotics [47] also shares some elements. 3SIL significantly differs from DAGGER and BC because it does not use an outside expert to obtain useful data, because of the stratified sampling procedure, and because of the selection of the shortest solutions for each problem in the training dataset. We include as an additional baseline an implementation of behavioral cloning (BC), where we regard proofs already encountered as coming from an expert. We minimize cross-entropy between the actions in proofs we have found and the predictions to train the predictor. For BC, there is no stratified sampling or shortest solution selection, only the minimization of cross-entropy between actions taken from recent successful solutions and the predictor's output.

Extensions. For the AIM tasks, we introduce two other techniques, *biased sampling* and *episode pruning*. In biased sampling, problems without a solution in the history are sampled 5 times more during episode collection than solved problems to accelerate progress. This was determined by testing 1, 2, 5 and 10 as sampling proportions. For episode pruning, when the agent encountered the same state twice, we prune the episode to exclude the looping before storing the episode. This helps the predictor learn to avoid these loops.

6 Neural Architectures

The tree-structured states representing expressions occurring during the tasks will be processed by a neural network. The neural network takes the tree-structured state and predicts an action to take that will bring the expression closer to being normalized or the theorem closer to being proven.

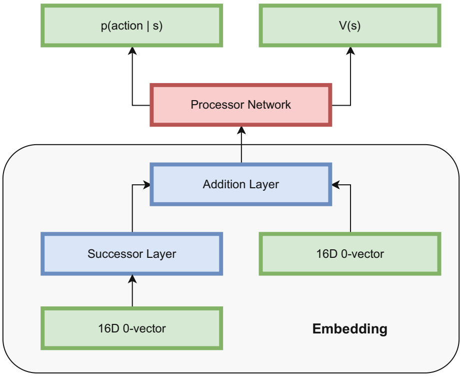


Fig. 2. Schematic representation of the creation of a representation of an expression (*an embedding*) using different neural network layers to represent different operations. The figure depicts the creation of a numerical representation for the Robinson arithmetic expression $(S(0) + 0)$. Note that the successor layer and the addition layer consist of trainable parameters, for which the values are set through gradient descent.

There are two main components to the neural network we use: an *embedding* tree neural network that outputs a numerical vector representing the tree-structured proof state and a second *processor* network that takes this vector representation of the state and outputs a distribution of the actions possible in the environment.[3]

Tree neural networks have been used in various settings, such as natural language processing [20] and also in Robinson arithmetic expression embedding [13]. These networks consist of smaller neural networks, each representing one of the possible functions that occur in the expressions. For example, there will be separate networks representing addition and multiplication. The cursor is a special unary operation node with its own network that we insert into the tree at the current location. For each unique constant, such as the constant 0 in RA or the identity element 1 for the AIM task, we generate a random vector (from a standard normal distribution) that will represent this leaf. In the case of the AIM task, these vectors are parameters that can be optimized during training.

At prediction time, the numerical representation of a tree is constructed by starting at the leaves of the tree, for which we can look up the generated vectors. These vectors act as input to the neural networks that represent the parent node's operation, yielding a new vector, which now represents the subtree of the parent node. The process repeats until there is a single vector for the entire tree after the root node is processed (see also Fig. 2).

The neural networks representing each operation consist of a linear transformation, a non-linearity in the form of a rectified linear unit (ReLU) and another linear transformation. In the case of binary operations, the first linear transformation will have an input dimension of $2n$ and an output dimension of n, where n is the dimension of the vectors representing leaves of the tree (the *internal representation size*). The weights representing these transformations are randomly initialized at the beginning of training.

When we have obtained a single vector embedding representing the entire tree data structure, this vector serves as the input to the *predictor* neural network, which consists of three linear layers, with non-linearities (Sigmoid/ReLU) in between these layers. The last layer has an output dimension equal to the number of possible actions in the environment. We obtain a probability distribution over the actions, e.g. by applying the softmax function to the output of this last layer. In the cases where we also need a value prediction, there is a parallel last layer that predicts the state's value (usually referred to as a *two-headed* network [41]). The internal representation size n for the Robinson arithmetic experiments is set to 16, for the AIM task this is 32. The number of neurons in each layer (except for the last one) of the predictor networks is 64.

In the AIM dataset task, an arbitrary number of variables can be introduced during the proof. These are represented by untrainable random vectors. We add a special neural network (with the same architecture as the networks representing unary operations, so from size n to n) that processes these vectors before they are

[3] In the reinforcement learning baselines that we use, this second *processor* network has the additional task of predicting the value of a state.

processed by the rest of the tree neural network embedding. The idea is that this neural network learns to project these new variable vectors into a subspace and that an arbitrary number of variables can be handled. The vectors are resampled at the start of each episode, so the agent cannot learn to recognize specific variables. This approach was partly inspired by the *prime* mechanism in [13], but we use separate vectors for all variables instead of building vectors sequentially. All our neural networks are implemented using the PyTorch library [31].

7 Experiments

We first describe our experiments on the Robinson arithmetic task, with which we designed the properties of our 3SIL approach with the help of comparisons with other algorithms. We then train a predictor using 3SIL on the AIMLEAP loop theory dataset, which we evaluate both as a standalone prover within the RL environment and as a neural guidance mechanism for the ATP Prover9.

7.1 Robinson Arithmetic Dataset

Dataset Details. The Robinson arithmetic dataset [11] is split into three distinct sets, based on the number of steps that it takes a fixed rewriting strategy to normalize the expression. This fixed strategy, LOPL, which stands for *left outermost proof length*, always rewrites the leftmost possible element. If it takes this strategy less than 90 steps to solve the problem, it is in the *low* difficulty category. Problems with a difficulty between 90 and 130 are in the *medium* category and a greater difficulty than 130 leads to the *high* category. The *high* dataset also contains problems the LOPL strategy could not solve within the time limit. The *low* dataset is split into a training and testing set. We train on the *low* difficulty problems, but after training we also test on problems with a higher difficulty. Because we have a difficulty measure for this dataset, we use a curriculum setup. We start by learning to normalize the expressions that a fixed strategy can normalize in a small amount of steps. This setup is similar to [11].

Training Setup. The 400 problems with the lowest difficulty are the starting point. Every time an agent reaches 95 percent success rate when evaluated on a sample of size 400 from these problems, we add 400 more difficult problems to set of training problems P. One iteration of the *collection* and *training* phase is called an *epoch.* Agents are evaluated after every epoch. The blocks of size 400 are called *levels.* The number of episodes m and f are set to 1000. For 3SIL and BC, the batch size BS is 32 and the number of batches NB is 250. The baselines are configured so that the number of episodes and training transitions is at least as many as the 3SIL/BC approaches. Episodes that take over 100 steps are stopped. ADAM [22] is used as an optimizer.

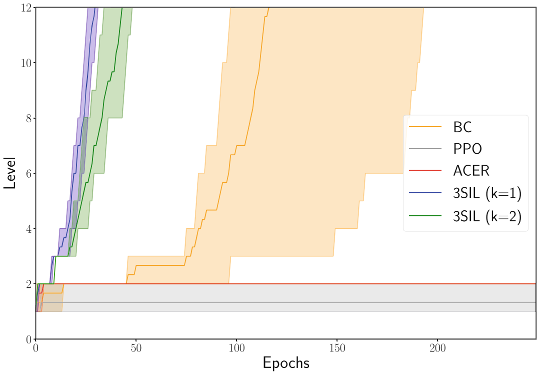


Fig. 3. The level in the curriculum reached by each method. Each method was run three times. The bold line shows the mean performance and the shaded region shows the minimum and maximum performance. K is the number of proofs stored per problem.

Results on RA Curriculum. In Fig. 3, we show the progression through the training curriculum for behavioral cloning (BC), the RL methods (PPO, ACER) and two configurations of 3SIL. Behavioral cloning simply imitates actions from successful episodes. Of the RL baselines, PPO reaches the second level in one run, while ACER steadily solves the first level and in the best run solves around 80% of the second level. Both methods do not learn enough solutions for the second level to advance to the third. A2C and SIL-PAAC do not reach the second level, so these are left out of the plot. However, they do learn to solve about 70–80% of the first 400 problems. From these results we can conclude that the RL baselines do not perform well on this task in our experiment. We attribute this to the difficulty of learning a good value function due to the sparse rewards (Sect. 5.1). Our hypothesis is that because this value estimate influences the policy updates, the RL methods do not learn well on this task. Note that the two methods with a trust region update mechanism, ACER and PPO, perform better than the methods without this mechanism. From these results, it is clear that 3SIL with 1 shortest proof stored, $k = 1$, is the best-performing configuration. It reaches the end of the training curriculum of about 5000 problems in 40 epochs. We experimented with $k = 3$ and $k = 4$, but these were both worse than $k = 2$.

Generalization. While our approach works well on the training set, we must check if the predictors generalize to unseen examples. Only the methods that reached the end of the curriculum are tested. In Table 1, we show the results of evaluating the performance of our predictors on the three different test sets: the unseen examples from the *low* dataset and the unseen examples from the *medium* and *high* datasets. Because we expect longer solutions, the episode limits are expanded from 100 steps to 200 and 250 for the *medium* and *high* datasets respectively. For the *low* and *medium* datasets, the second of which contains problems with more difficult solutions than the training data, the predictors solve almost all test problems. For the *high* difficulty dataset, the performance drops by at least 20% points. Our method outperforms the Monte Carlo Tree Search approach used in [11] on the same datasets, which got to 0.954 on the *low* dataset with 1600 iterations and 0.786 on the *medium* dataset (no results on the *high* dataset were reported). These results indicate that this training method might be strong enough to perform well on the AIM rewriting RL task.

Table 1. Generalization with greedy evaluation on the test set for the Robinson arithmetic normalization tasks, shown as average success rate and standard deviation from 3 training runs. Generalization is high on the low and medium difficulty (training data is similar to the low difficulty dataset). With high difficulty data, performance drops.

	Low	Medium	High
3SIL ($\kappa = 1$)	1.00 ± 0.01	0.98 ± 0.03	0.77 ± 0.10
3SIL ($\kappa = 2$)	0.99 ± 0.00	0.96 ± 0.01	0.66 ± 0.08
BC	0.98 ± 0.01	0.98 ± 0.01	0.56 ± 0.05

7.2 AIM Conjecture Dataset

Training Setup. Finally, we train and evaluate 3SIL on the AIM Conjecture dataset. We apply 3SIL ($k = 1$) to train predictors in the AIMLEAP environment. Ten percent of the AIM dataset is used as a hold-out test set, not seen during training. As there is no estimate for the difficulty of the problems in terms of the actions available to the predictor, we do not use a curriculum ordering for these experiments. The number m of episodes collected before training is set to 2,000,000. These random proof attempts result in about 300 proofs. The predictor learns from these proofs and afterwards the search for new proofs is also guided by its predictions. For the AIM experiments, episodes are stopped after 30 steps in the AIMLEAP environment. The predictors are trained for 100 epochs. The number of collected episodes per epoch f is 10,000. The successful proofs are stored, and the shortest proof for each theorem is kept. NB is 500 and BS is set to 32. The number of problems with a solution in the history after each epoch of the training run is shown in Fig. 4.

Results as a Standalone Prover. After 100 epochs, about 2500 of 3114 problems in the training dataset have a solution in their history. To test the generalization capability of the predictors, we inspect their performance on the holdout test set problems. In Table 2 we compare the success rate of the trained predictors on the holdout test set with three different automated theorem provers: E [37,38], Waldmeister [19] and Prover9. E is currently one of the best overall automated theorem provers [44], Waldmeister is a prover specialized in memory-efficient equational theorem proving [18] and Prover9 is the theorem prover that

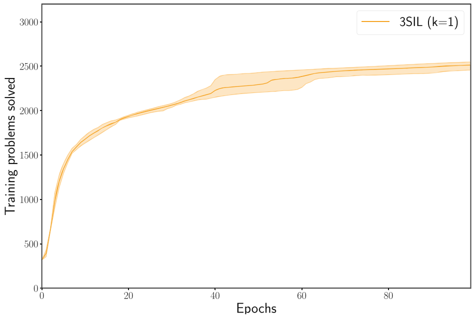


Fig. 4. The number of training problems for which a solution was encountered and stored (cumulative). At the start of the training, the models rapidly collect more solutions, but after 100 epochs, the process slows down and settles at about 2500 problems with known solutions. The minimum, maximum and mean of three runs are shown.

is used for AIM conjecture research and the prover that the dataset was generated by. Waldmeister and E are the best performing solvers in competitions for the relevant unit equality (UEQ) category [44].

Table 2. Theorem proving performance on the hold-out test set in fraction of problems solved. Means and standard deviations are the results of evaluations of 3 different predictors from 3 different training runs on the 354 unseen test set problems.

Method	Success Rate
Prover9 (60 s)	0.833
E (60 s)	0.802
Predictor + AIMLEAP(60 s)	0.702 ± 0.015
Waldmeister (60 s)	0.655
Predictor + AIMLEAP (1×)	0.586 ± 0.029

The results show that a single greedy evaluation of the predictor trying to solve the problem in the AIMLEAP environment is not as strong as the theorem proving software. However, the theorem provers got 60 s of execution time, and the execution of the predictor, including interaction with AIMLEAP, takes on average less than 1 s. We allowed the predictor setup to use 60 s, by running attempts in AIMLEAP until the time was up, sampling actions from the predictor's distribution with 5% noise, instead of using greedy execution. With this approach, the predictor setup outperforms Waldmeister.[4] Figure 5 shows the overlap between the problems solved by each prover. The diagram shows that each theorem prover found a few solutions that no other prover could find within

[4] After the initial experiments, we also evaluated Twee [42], which won the most recent UEQ track: it can prove most of the test problems in 60 s, only failing for 1 problem.

the time limit. Almost half of all problems from the test set that are solved are solved by all four systems.

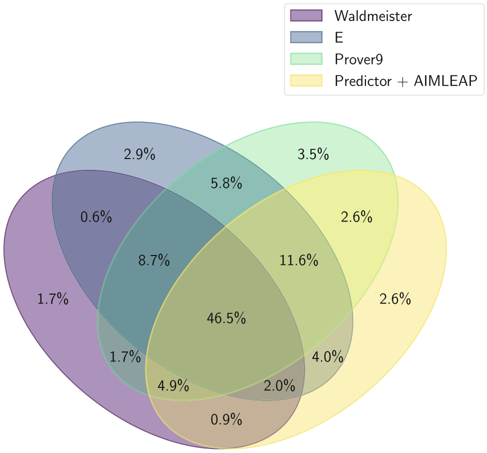


Fig. 5. Venn diagram of the test set problems solved by each solver with 60 s time limit.

Results of Neural Rewriting Combined with Prover9. We also combine the predictor with *Prover9*. In this setup, the predictor modifies the starting form of the goal, for a maximum of 1 s in the AIMLEAP environment. This produces new expressions on one or both sides of the equality. We then add, as lemmas, equalities between the left-hand side of the goal before the predictor's rewriting and after each rewriting (see Fig. 1). The same is done for the right-hand side. For each problem, this procedure yields new lemmas that are added to the problem specification file that is given to *Prover9*.

Table 3. Prover9 theorem proving performance on the hold-out test set when injecting lemmas suggested by the learned predictor. *Prover9*'s performance increases when using the suggested lemmas.

Method	Success Rate
Prover9 (1 s)	0.715
Prover9 (2 s)	0.746
Prover9 (60 s)	0.833
Rewriting (1 s) + Prover9 (1 s)	0.841 ± 0.019
Rewriting (1 s) + Prover9 (59 s)	**0.902** ± 0.016

In Table 3, it is shown that adding lemmas suggested by the rewriting actions of the trained predictor improves the performance of *Prover9*. Running *Prover9* for 2 s results in better performance than running it for 1 s, as expected. The combined (1 s + 1 s) system improved on *Prover9's* 2-s performance by 12.7% (= 0.841/0.746), indicating that the predictor suggests useful lemmas. Additionally, 1 s of neural rewriting combined with 59 s of Prover9 search proves almost 8.3% (= 0.902/0.833) more theorems than Prover9 with a 60 s time limit (Table 2).

7.3 Implementation Details

All experiments for the Robinson task were run on a 16 core Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60 GHz. The AIM experiments were run on a 72 core Intel(R) Xeon(R) Gold 6140 CPU @ 2.30 GHz. All calculations were done on CPU. The PPO implementation was adapted from an existing implementation [3]. The model was updated every 2000 timesteps, the PPO clip coefficient was set to 0.2. The learning rate was 0.002 and the discount factor γ was set to 0.99. The ACER implementation was adapted from an available implementation [8]. The replay buffer size was 20,000. The truncation parameter was 10 and the model was updated every 100 steps. The replay ratio was set to 4. Trust region decay was set to 0.99 and the constraint was set to 1. The discount factor was set to 0.99 and the learning rate to 0.001. Off-policy minibatch size was set to 1. The A2C and SIL implementations were based on Pytorch actor-critic example code available at the PyTorch repository [33]. For the A2C algorithm, we experimented with two formulations of the advantage function: the 1-step lookahead estimate $(r_t + \gamma\mathcal{V}_\mu(s_{t+1})) - \mathcal{V}_\mu(s_t)$ and the $R_t - \mathcal{V}_\mu(s_t)$ formulation. However, we did not observe different performance, so we opted in the end for the 1-step estimate favored in the original A2C publication. For SIL-PAAC, we implemented the SIL loss on top of the A2C implementation. There is also a prioritized replay buffer with an exponent of 0.6, as in the original paper. Each epoch, 8000 (250 batches of size 32) transitions were taken from the prioritized replay buffer in the SIL step of the algorithm. The size of the prioritized replay buffer was 40,000. The critic loss weight was set to 0.01 as in the original paper. For the 3SIL and behavioral cloning implementations, we sample 8000 transitions (250 batches of size 32) from the replay buffer or history. For the behavioral cloning, we used a buffer of size 40,000. An example implementation of 3SIL can be found in the RewriteRL repository. On the Robinson arithmetic task, for 3SIL and BC, the evaluation is done greedily (always take the highest probability actions). For the other methods, we performed experiments with both greedy and non-greedy (sample from the predictor distribution and add 5% noise) evaluation and show the results the best-performing setting (which in most cases was the non-greedy evaluation, except for PPO). On the AIM task, we evaluate greedily with 3SIL.

AIMLEAP expects a distance estimate for each applicable action. This represents the estimated distance to a proof. This behavior was converted to a reinforcement learning setup by always setting the chosen action of the model to the minimum distance and all other actions to a distance larger than the maximum proof length. Only the chosen action is then carried out.

Versions of the automated theorem provers used: Version 2.5 of E [39], the Nov 2017 version of Prover9 [26] and the Feb 2018 version of Waldmeister [46] and version 2.4.1 of Twee [43].

8 Conclusion and Future Work

Our experiments show that a neural rewriter, trained with the 3SIL method that we designed, can learn to suggest useful lemmas that assist an ATP and improve its proving performance. With the same limit of 1 min, Prover9 managed to prove close to 8.3% more theorems. Furthermore, our 3SIL training method is powerful enough to train an equational prover from zero knowledge that can compete with hand-engineered provers, such as Waldmeister. Our system on its own proves 70.2% of the unseen test problems in 60s, while Waldmeister proved 65.5%.

In future work, we will apply our method to other equational reasoning tasks. An especially interesting research direction concerns selecting which proofs to learn from: some sub-proofs might be more general than other sub-proofs. The incorporation of graph neural networks instead of tree neural networks may improve the performance of the predictor, since in graph neural networks information not only propagates from the leaves to the root, but also through all other connections.

Acknowledgements. We would like to thank Chad Brown for his work with the AIMLEAP software. In addition, we thank Thibault Gauthier and Bartosz Piotrowski for their help with the Robinson arithmetic rewriting task and the AIM rewriting task respectively. We also thank the referees of the IJCAR conference for their useful comments.

This work was partially supported by the European Regional Development Fund under the Czech project AI&Reasoning no. CZ.02.1.01/0.0/0.0/15_003/ 0000466 (JP, JU), Amazon Research Awards (JP, JU) and by the Czech MEYS under the ERC CZ project *POSTMAN* no. LL1902 (JP, MJ).

This article is part of the RICAIP project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 857306.

References

1. Alama, J., Heskes, T., Kühlwein, D., Tsivtsivadze, E., Urban, J.: Premise selection for mathematics by corpus analysis and kernel methods. J. Autom. Reason. **52**(2), 191–213 (2013). https://doi.org/10.1007/s10817-013-9286-5
2. Bansal, K., Loos, S., Rabe, M., Szegedy, C., Wilcox, S.: HOList: an environment for machine learning of higher order logic theorem proving. In: International Conference on Machine Learning, pp. 454–463 (2019)
3. Barhate, N.: Implementation of PPO algorithm. https://github.com/nikhilbarhate99

4. Berner, C., et al.: DOTA 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680 (2019)
5. Blaauwbroek, L., Urban, J., Geuvers, H.: The Tactician. In: Benzmüller, C., Miller, B. (eds.) CICM 2020. LNCS (LNAI), vol. 12236, pp. 271–277. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53518-6_17
6. Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. J. Formalized Reason. **9**(1), 101–148 (2016). https://doi.org/10.6092/issn.1972-5787/4593
7. Brown, C.E., Piotrowski, B., Urban, J.: Learning to advise an equational prover. Artif. Intell. Theorem Proving, 1–13 (2020)
8. Chételat, D.: Implementation of ACER algorithm. https://github.com/dchetelat/acer
9. Chvalovský, K., Jakubův, J., Suda, M., Urban, J.: ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 197–215. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_12
10. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
11. Gauthier, T.: Deep reinforcement learning in HOL4. arXiv preprint arXiv:1910.11797v1 (2019)
12. Gauthier, T.: Deep reinforcement learning for synthesizing functions in higher-order logic. In: International Conference on Logic for Programming, Artificial Intelligence and Reasoning (2020)
13. Gauthier, T.: Tree neural networks in HOL4. In: Benzmüller, C., Miller, B. (eds.) CICM 2020. LNCS (LNAI), vol. 12236, pp. 278–283. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53518-6_18
14. Gauthier, T., Kaliszyk, C., Urban, J., Kumar, R., Norrish, M.: TacticToe: learning to prove with tactics. J. Autom. Reason. **65**, 1–30 (2020)
15. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 369–376 (2006)
16. He, H., Daume, H., III., Eisner, J.M.: Learning to search in branch and bound algorithms. Adv. Neural Inf. Process. Syst. **27**, 3293–3301 (2014)
17. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
18. Hillenbrand, T., Buch, A., Vogt, R., Löchner, B.: WALDMEISTER - high-performance equational deduction. J. Autom. Reasoning **18**, 265–270 (2004)
19. Hillenbrand, T.: Citius altius fortius: lessons learned from the theorem prover Waldmeister. ENTCS **86**(1), 9–21 (2003)
20. Irsoy, O., Cardie, C.: Deep recursive neural networks for compositionality in language. Adv. Neural Inf. Process. Syst. **27**, 2096–2104 (2014)
21. Kaliszyk, C., Urban, J., Michalewski, H., Olšák, M.: Reinforcement learning of theorem proving. Adv. Neural Inf. Process. Syst. **31**, 8822–8833 (2018)
22. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
23. Kinyon, M.: Proof simplification and automated theorem proving. CoRR abs/1808.04251 (2018). http://arxiv.org/abs/1808.04251

24. Kinyon, M., Veroff, R., Vojtěchovský, P.: Loops with abelian inner mapping groups: an application of automated deduction. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics. LNCS (LNAI), vol. 7788, pp. 151–164. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36675-8_8
25. McCune, W.: Prover9 and Mace (2010). http://www.cs.unm.edu/~mccune/prover9/
26. McCune, W.: Prover9. https://github.com/ai4reason/Prover9
27. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937 (2016)
28. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
29. Oh, J., Guo, Y., Singh, S., Lee, H.: Self-imitation learning. In: International Conference on Machine Learning, pp. 3878–3887 (2018)
30. Overbeek, R.A.: A new class of automated theorem-proving algorithms. J. ACM **21**(2), 191–200 (1974). https://doi.org/10.1145/321812.321814
31. Paszke, A., et al.: PyTorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 32, pp. 8024–8035. Curran Associates, Inc. (2019). http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
32. Phillips, J., Stanovský, D.: Automated theorem proving in quasigroup and loop theory. AI Commun. **23**(2–3), 267–283 (2010)
33. PyTorch: RL Examples. https://github.com/pytorch/examples/tree/main/reinforcement_learning
34. Ross, S., Gordon, G., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, pp. 627–635 (2011)
35. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 1889–1897. PMLR, Lille (2015). https://proceedings.mlr.press/v37/schulman15.html
36. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
37. Schulz, S.: E - a brainiac theorem prover. AI Commun. **15**(2–3), 111–126 (2002)
38. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, Higher, Stronger: E 2.3. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 495–507. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_29
39. Schulz, S.: Eprover. https://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html
40. Silver, D.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
41. Silver, D., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
42. Smallbone, N.: Twee: an equational theorem prover. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 602–613. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79876-5_35
43. Smallbone, N.: Twee 2.4.1. https://github.com/nick8325/twee/releases/download/2.4.1/twee-2.4.1-linux-amd64
44. Sutcliffe, G.: The CADE-27 automated theorem proving system competition - CASC-27. AI Commun. **32**(5–6), 373–389 (2020)

45. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge (2018)
46. Hillenbrand, T., Buch, A., Vogt, R., Löchner, B.: Waldmeister (2022). https://www.mpi-inf.mpg.de/departments/automation-of-logic/software/waldmeister/download
47. Torabi, F., Warnell, G., Stone, P.: Behavioral cloning from observation. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI 2018, pp. 4950–4957. AAAI Press (2018)
48. Veroff, R.: Using hints to increase the effectiveness of an automated reasoning program: Case studies. J. Autom. Reason. **16**(3), 223–239 (1996). https://doi.org/10.1007/BF00252178
49. Wang, Z., et al.: Sample efficient actor-critic with experience replay. In: International Conference on Learning Representations (2016)

Rensets and Renaming-Based Recursion for Syntax with Bindings

Andrei Popescu(✉)

Department of Computer Science, University of Sheffield, Sheffield, UK
a.popescu@sheffield.ac.uk

Abstract. I introduce *renaming-enriched sets* (*rensets* for short), which are algebraic structures axiomatizing fundamental properties of renaming (also known as variable-for-variable substitution) on syntax with bindings. Rensets compare favorably in some respects with the well-known foundation based on nominal sets. In particular, renaming is a more fundamental operator than the nominal swapping operator and enjoys a simpler, equationally expressed relationship with the variable-freshness predicate. Together with some natural axioms matching properties of the syntactic constructors, rensets yield a truly minimalistic characterization of λ-calculus terms as an abstract datatype – one involving an infinite set of *unconditional equations*, referring only to the most fundamental term operators: the constructors and renaming. This characterization yields a recursion principle, which (similarly to the case of nominal sets) can be improved by incorporating Barendregt's variable convention. When interpreting syntax in semantic domains, my renaming-based recursor is easier to deploy than the nominal recursor. My results have been validated with the proof assistant Isabelle/HOL.

1 Introduction

Formal reasoning about syntax with bindings is necessary for the meta-theory of logics, calculi and programming languages, and is notoriously error-prone. A great deal of research has been put into formal frameworks that make the specification of, and the reasoning about bindings more manageable.

Researchers wishing to formalize work involving syntax with bindings must choose a paradigm for representing and manipulating syntax—typically a variant of one of the "big three": nameful (sometimes called "nominal" reflecting its best known incarnation, nominal logic [23,39]), nameless (De Bruijn) [4,13,49,51] and higher-order abstract syntax (HOAS) [19,20,28,34,35]. Each paradigm has distinct advantages and drawbacks compared with each of the others, some discussed at length, e.g., in [1,9] and [25, §8.5]. And there are also hybrid approaches, which combine some of the advantages [14,18,42,47].

A significant advantage of the nameful paradigm is that it stays close to the way one informally defines and manipulates syntax when describing systems in textbooks and research papers—where the binding variables are explicitly

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 618–639, 2022.
https://doi.org/10.1007/978-3-031-10769-6_36

indicated. This can in principle ensure transparency of the formalization and allows the formalizer to focus on the high-level ideas. However, it only works if the technical challenge faced by the nameful paradigm is properly addressed: enabling the seamless definition and manipulation of concepts "up to alpha-equivalence", i.e., in such a way that the names of the bound variables are (present but nevertheless) inconsequential. This is particularly stringent in the case of recursion due to the binding constructors of terms not being free, hence not being *a priori* traversable recursively—in that simply writing some recursive clauses that traverse the constructors is not *a priori* guaranteed to produce a correct definition, but needs certain favorable conditions. The problem has been addressed by researchers in the form of tailored *nameful recursors* [23,33,39,43,56,57], which are theorems that identify such favorable conditions and, based on them, guarantee the existence of functions that recurse over the non-free constructors.

In this paper, I make a contribution to the nameful paradigm in general, and to nameful recursion in particular. I introduce *rensets*, which are algebraic structures axiomatizing the properties of renaming, also known as variable-for-variable substitution, on terms with bindings (Sect. 3). Rensets differ from nominal sets (Sect. 2.2), which form the foundation of nominal logic, by their focus on (not necessarily injective) renaming rather than swapping (or permutation). Similarly to nominal sets, rensets are pervasive: Not only do the variables and terms form rensets, but so do any container-type combinations of rensets.

While lacking the pleasant symmetry of swapping, my axiomatization of renaming has its advantages. First, renaming is more fundamental than swapping because, at an abstract axiomatic level, renaming can define swapping but not vice versa (Sect. 4). The second advantage is about the ability to define another central operator: the variable freshness predicate. While the definability of freshness from swapping is a signature trait of nominal logic, my renaming-based alternative fares even better: In rensets freshness has a simple, first-order definition (Sect. 3). This contrasts the nominal logic definition, which involves a second-order statement about (co)finiteness of a set of variables. The third advantage is largely a consequence of the second: Rensets enriched with constructor-like operators facilitate an equational characterization of terms with bindings (using an infinite set of unconditional equations), which does not seem possible for swapping (Sect. 5.1). This produces a recursion principle (Sect. 5.2) which, like the nominal recursor, caters for Barendregt's variable convention, and in some cases is easier to apply than the nominal recursor—for example when interpreting syntax in semantic domains (Sect. 5.3).

In summary, I argue that my renaming-based axiomatization offers some benefits that strengthen the arsenal of the nameful paradigm: a simpler representation of freshness, a minimalistic equational characterization of terms, and a convenient recursion principle. My results are established with high confidence thanks to having been mechanized in Isabelle/HOL [32]. The mechanization is available [44] from Isabelle's Archive of Formal Proofs.

Here is the structure of the rest of this paper: Sect. 2 provides background on terms with bindings and on nominal logic. Section 3 introduces rensets and describes their basic properties. Section 4 establishes a formal connection to nominal sets. Section 5 discusses substitutive-set-based recursion. Section 6 discusses related work. A technical report [45] associated to this paper includes an appendix with more examples and results and more background on nominal sets.

2 Background

This section recalls the terms of λ-calculus and their basic operators (Sect. 2.1), and aspects of nominal logic including nominal sets and nominal recursion (Sect. 2.2).

2.1 Terms with Bindings

I work with the paradigmatic syntax of (untyped) λ-calculus. However, my results generalize routinely to syntaxes specified by arbitrary binding signatures such as the ones in [22, §2], [39,59] or [12].

Let Var be a countably infinite set of variables, ranged over by x, y, z etc. The set Trm of *λ-terms* (or *terms* for short), ranged over by t, t_1, t_2 etc., is defined by the grammar $t ::= \ \mathsf{Vr}\ x \ \mid\ \mathsf{Ap}\ t_1\ t_2 \ \mid\ \mathsf{Lm}\ x\ t$
with the proviso that terms are equated (identified) modulo alpha-equivalence (also known as naming equivalence). Thus, for example, if $x \neq z \neq y$ then $\mathsf{Lm}\ x\ (\mathsf{Ap}\ (\mathsf{Vr}\ x)\ (\mathsf{Vr}\ z))$ and $\mathsf{Lm}\ y\ (\mathsf{Ap}\ (\mathsf{Vr}\ y)\ (\mathsf{Vr}\ z))$ are considered to be the same term. I will often omit Vr when writing terms, as in, e.g., $\mathsf{Lm}\ x\ x$.

What the above specification means is (something equivalent to) the following: One first defines the set PTrm of *pre-terms* as freely generated by the grammar $p ::= \ \mathsf{PVr}\ x \ \mid\ \mathsf{PAp}\ p_1\ p_2 \ \mid\ \mathsf{PLm}\ x\ p$. Then one defines the alpha-equivalence relation $\equiv\ : \mathsf{PTrm} \to \mathsf{PTrm} \to \mathsf{Bool}$ inductively, proves that it is an equivalence, and defines Trm by quotienting PTrm to alpha-equivalence, i.e., $\mathsf{Trm} = \mathsf{PTrm}/\equiv$. Finally, one proves that the pre-term constructors are compatible with $\equiv$, and defines the term counterpart of these constructors: $\mathsf{Vr} : \mathsf{Var} \to \mathsf{Trm}$, $\mathsf{Ap} : \mathsf{Trm} \to \mathsf{Trm} \to \mathsf{Trm}$ and $\mathsf{Lm} : \mathsf{Var} \to \mathsf{Trm} \to \mathsf{Trm}$.

The above constructions are technical, but well-understood, and can be fully automated for an arbitrary syntax with bindings (not just that of λ-calculus); and tools such as the Isabelle/Nominal package [59,60] provide this automation, hiding pre-terms completely from the end user. In formal and informal presentations alike, one usually prefers to forget about pre-terms, and work with terms only. This has several advantages, including (1) being able to formalize concepts at the right abstraction level (since in most applications the naming of bound variables should be inconsequential) and (2) the renaming operator being well-behaved. However, there are some difficulties that need to be overcome when working with terms, and in this paper I focus on one of the major ones: providing recursion principles, i.e., mechanisms for defining functions by recursing over terms. This difficulty arises essentially because, unlike in the case of pre-term constructors, the binding constructor for terms is not free.

The main characters of my paper will be (generalizations of) some common operations and relations on Trm, namely:

- the constructors $\mathsf{Vr} : \mathsf{Var} \to \mathsf{Trm}$, $\mathsf{Ap} : \mathsf{Trm} \to \mathsf{Trm} \to \mathsf{Trm}$ and $\mathsf{Lm} : \mathsf{Var} \to \mathsf{Trm} \to \mathsf{Trm}$
- (capture-avoiding) renaming, also known as (capture-avoiding) substitution of variables for variables $\_[\_/\_] : \mathsf{Trm} \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Trm}$; e.g., we have $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ [x/y] = \mathsf{Lm}\ x'\ (\mathsf{Ap}\ x'\ x)$
- swapping $\_[\_\wedge\_] : \mathsf{Trm} \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Trm}$; e.g., we have $(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ [x\wedge y] = \mathsf{Lm}\ y\ (\mathsf{Ap}\ y\ x)$
- the free-variable operator $\mathsf{FV} : \mathsf{Trm} \to \mathsf{Pow}(\mathsf{Var})$ (where $\mathsf{Pow}(\mathsf{Var})$ is the powerset of Var); e.g., we have $\mathsf{FV}(\mathsf{Lm}\ x\ (\mathsf{Ap}\ y\ x)) = \{y\}$
- freshness $\_\#\_ : \mathsf{Var} \to \mathsf{Trm} \to \mathsf{Bool}$; e.g., we have $x \,\#\, (\mathsf{Lm}\ x\ x)$; and assuming $x \neq y$, we have $\neg\, x \,\#\, (\mathsf{Lm}\ y\ x)$

The free-variable and freshness operators are of course related: A variable x is fresh for a term t (i.e., $x \,\#\, t$) if and only if it is not free in t (i.e., $x \notin \mathsf{FV}(t)$). The renaming operator $\_[\_/\_] : \mathsf{Trm} \to \mathsf{Var} \to \mathsf{Var} \to \mathsf{Trm}$ substitutes (in terms) *variables* for variables, not terms for variables. (But an algebraization of term-for-variable substitution is discussed in [45, Appendix D].)

2.2 Background on Nominal Logic

I will employ a formulation of nominal logic [38,39,57] that does not require any special logical foundation, e.g., axiomatic nominal set theory. For simplicity, I prefer the swapping-based formulation [38] to the equivalent permutation-based formulation—[45, Appendix C] gives details on these two alternatives.

A *pre-nominal set* is a pair $\mathcal{A} = (A, \_[\_\wedge\_])$ where A is a set and $\_[\_\wedge\_] : A \to \mathsf{Perm} \to A$ is a function called *the swapping operator of* $\mathcal{A}$ satisfying the following properties for all $a \in A$ and $x, x_1, x_2, y_1, y_2 \in \mathsf{Var}$:

$$\begin{aligned} \text{Identity:}\quad & a[x\wedge x] = a \\ \text{Involution:}\quad & a[x_1\wedge x_2][x_1\wedge x_2] = a \\ \text{Compositionality:}\quad & a[x_1\wedge x_2][y_1\wedge y_2] = a[y_1\wedge y_2][(x_1[y_1\wedge y_2])\wedge(x_2[y_1\wedge y_2])] \end{aligned}$$

Given a pre-nominal set $\mathcal{A} = (A, \_[\_\wedge\_])$, an element $a \in A$ and a set $X \subseteq \mathsf{Var}$, one says that *a is supported by X* if $a[x\wedge y] = a$ holds for all $x, y \in \mathsf{Var}$ such that $x, y \notin X$. An element $a \in A$ is called *finitely supported* if there exists a finite set $X \subseteq A$ such that a is supported by X. A *nominal set* is a pre-nominal set $\mathcal{A} = (A, \_[\_\wedge\_])$ such that every element of a is finitely supported. If $\mathcal{A} = (A, \_[\_\wedge\_])$ is a nominal set and $a \in A$, then the smallest set $X \subseteq A$ such that a is supported by X exists, and is denoted by $\mathsf{supp}^{\mathcal{A}}\, a$ and called the *support of a*. One calls a variable x *fresh for a*, written $x \,\#\, a$, if $x \notin \mathsf{supp}^{\mathcal{A}}\, a$.

An alternative, more direct definition of freshness (which is preferred, e.g., by Isabelle/Nominal [59,60]) is provided by the following proposition:

Proposition 1. For any nominal set $\mathcal{A} = (A, \_[\_\wedge\_])$ and any $x \in \mathsf{Var}$ and $a \in A$, it holds that $x \,\#\, a$ if and only if the set $\{y \mid a[y\wedge x] \neq a\}$ is finite.

Given two pre-nominal sets $\mathcal{A} = (A, \_[\_\wedge\_])$ and $\mathcal{B} = (B, \_[\_\wedge\_])$, the set $F = (A \to B)$ of functions from A to B becomes a pre-nominal set $\mathcal{F} = (F, \_[\_\wedge\_])$ by defining $f[x\wedge y]$ to send each $a \in A$ to $(f(a[x\wedge y]))[x\wedge y]$. $\mathcal{F}$ is not a nominal set because not all functions are finitely supported (though of course one obtains a nominal set by restricting to finitely supported functions).

The set of terms together with their swapping operator, $(\mathsf{Trm}, \_[\_\wedge\_])$, forms a nominal set, where the support of a term is precisely its set of free variables. However, the power of nominal logic resides in the fact that not only the set of terms, but also many other sets can be organized as nominal sets—including the target domains of many functions one may wish to define on terms. This gives rise to a convenient mechanism for defining functions recursively on terms:

Theorem 2 [39]**.** Let $\mathcal{A} = (A, \_[\_])$ be a nominal set and let $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$ and $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$ be some functions, all supported by a finite set X of variables and with $\mathsf{Lm}^{\mathcal{A}}$ satisfying the following freshness condition for binders (FCB): There exists $x \in \mathsf{Var}$ such that $x \notin X$ and $x \# \mathsf{Lm}^{\mathcal{A}}\ x\ a$ for all $a \in A$.

Then there exists a unique function $f : \mathsf{Trm} \to A$ that is supported by X and such that the following hold for all $x \in \mathsf{Var}$ and $t_1, t_2, t \in \mathsf{Trm}$:

(i) $f\,(\mathsf{Vr}\ x) = \mathsf{Vr}^{\mathcal{A}}\ x$ (ii) $f\,(\mathsf{Ap}\ t_1\ t_2) = \mathsf{Ap}^{\mathcal{A}}\ (f\ t_1)\ (f\ t_2)$

(iii) $f\,(\mathsf{Lm}\ x\ t) = \mathsf{Lm}^{\mathcal{A}}\ x\ (f\ t)$ if $x \notin X$

A useful feature of nominal recursion is the support for Barendregt's famous *variable convention* [8, p. 26]: "If [the terms] $t_1, \ldots, t_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables." The above recursion principle adheres to this convention by fixing a finite set X of variables meant to be free in the definition context and guaranteeing that the bound variables in the definitional clauses are distinct from them. Formally, the target domain operators $\mathsf{Vr}^{\mathcal{A}}$, $\mathsf{Ap}^{\mathcal{A}}$ and $\mathsf{Lm}^{\mathcal{A}}$ are supported by X, and the clause for λ-abstraction is conditioned by the binding variable x being outside of X. (The Barendregt convention is also present in nominal logic via induction principles [39,58–60].)

3 Rensets

This section introduces rensets, an alternative to nominal sets that axiomatize renaming rather than swapping or permutation.

A *renaming-enriched set* (*renset* for short) is a pair $\mathcal{A} = (A, \_[\_/\_])$ where A is a set and $\_[\_/\_] : A \to \mathsf{Var} \to \mathsf{Var} \to A$ is an operator such that the following hold for all $x, x_1, x_2, x_3, y, y_1, y_2 \in \mathsf{Var}$ and $a \in A$:

Identity:	$a[x/x] = a$
Idempotence:	If $x_1 \neq y$ then $a[x_1/y][x_2/y] = a[x_1/y]$
Chaining:	If $y \neq x_2$ then $a[y/x_2][x_2/x_1][x_3/x_2] = a[y/x_2][x_3/x_1]$
Commutativity:	If $x_2 \neq y_1 \neq x_1 \neq y_2$ then $a[x_2/x_1][y_2/y_1] = a[y_2/y_1][x_2/x_1]$

Let us call A the *carrier* of $\mathcal{A}$ and $\_[\_/\_]$ the *renaming operator* of $\mathcal{A}$. Similarly to the case of terms, we think of the elements $a \in A$ as some kind of variable-bearing entities and of $a[y/x]$ as the result of substituting x with y in a. With this intuition, the above properties are natural: Identity says that substituting a variable with itself has no effect. Idempotence acknowledges the fact that, after its renaming, a variable y is no longer there, so substituting it again has no effect. Chaining says that a chain of renamings $x_3/x_2/x_1$ has the same effect as the end-to-end renaming x_3/x_1 provided there is no interference from x_2, which is ensured by initially substituting x_2 with some other variable y. Finally, Commutativity allows the reordering of any two independent renamings.

Examples. $(\mathsf{Var}, \_[\_/\_])$ and $(\mathsf{Trm}, \_[\_/\_])$, the sets of variables and terms with the standard renaming operator on them, form rensets. Moreover, given any functor F on the category of sets and a renset $\mathcal{A} = (A, \_[\_/\_])$, let us define the renset $F\,\mathcal{A} = (F\,A, \_[\_/\_])$ as follows: for any $k \in F\,A$ and $x, y \in \mathsf{Var}$, $k[x/y] = F\,(\_[x/y])\,k$, where the last occurrence of F refers to the action of the functor on morphisms. This means that one can freely build new rensets from existing ones using container types (which are particular kinds of functors)—e.g., lists, sets, trees etc. Another way to put it: Rensets are closed under datatype and codatatype constructions [55].

In what follows, let us fix a renset $\mathcal{A} = (A, \_[\_/\_])$. One can define the notion of freshness of a variable for an element of a in the style of nominal logic. But the next proposition shows that simpler formulations are available.

Proposition 3. The following are equivalent:
(1) The set $\{y \in \mathsf{Var} \mid a[y/x] \neq a\}$ is finite.
(2) $a[y/x] = a$ for all $y \in \mathsf{Var}$. (3) $a[y/x] = a$ for some $y \in \mathsf{Var} \smallsetminus \{x\}$.

Let us define the predicate $\_\#\_ : \mathsf{Var} \to A \to \mathsf{Bool}$ as follows: $x \# a$, read x *is fresh for* a, if either of Proposition 3's equivalent properties holds.

Thus, points (1)–(3) above are three alternative formulations of $x \# a$, all referring to the lack of effect of substituting y for x, expressed as $a[y/x] = a$: namely that this phenomenon affects (1) all but a finite number of variables y, (2) all variables y, or (3) some variable $y \neq x$. The first formulation is the most complex of the three—it is the nominal definition, but using renaming instead of swapping. The other two formulations do not have counterparts in nominal logic, essentially because swapping is not as "efficient" as renaming at exposing freshness. In particular, (3) does not have a nominal counterpart because there is no single-swapping litmus test for freshness. The closest we can get to property (3) in a nominal set is the following: x is fresh for a if and only $a[y \wedge x] = a$ holds for some fresh y—but this needs freshness to explain freshness!

Examples (continued). For the rensets of variables and terms, freshness defined as above coincides with the expected operators: distinctness in the case of variables and standard freshness in the case of terms. And applying the definition of freshness to rensets obtained using finitary container types has similarly intuitive outcomes; for example, the freshness of a variable x for a list of items $[a_1, \ldots, a_n]$ means that x is fresh for each item a_i in the list.

Freshness satisfies some intuitive properties, which can be easily proved from its definition and the renset axioms. In particular, point (2) of the next proposition is the freshness-based version of the Chaining axiom.

Proposition 4. The following hold:
(1) If $x \# a$ then $a[y/x] = a$ (2) $x_2 \# a$ then $a[x_2/x_1][x_3/x_2] = a[x_3/x_1]$
(3) If $z \# a$ or $z = x$, and $x \# a$ or $z \neq y$, then $z \# a[y/x]$

4 Connection to Nominal Sets

So far I focused on consequences of the purely equational theory of rensets, without making any assumption about cardinality. But after additionally postulating a nominal-style finite support property, one can show that rensets give rise to nominal sets—which is what I will do in this section.

Let us say that a renset $\mathcal{A} = (A, \_[\_/\_])$ has the *Finite Support* property if, for all $a \in A$, the set $\{x \in \mathsf{Var} \mid \neg\, x \# a\}$ is finite.

Let $\mathcal{A} = (A, \_[\_/\_])$ be a renset satisfying Finite Support. Let us define the swapping operator $\_[\_ \wedge \_] : A \to \mathsf{Var} \to \mathsf{Var} \to A$ as follows: $a[x_1 \wedge x_2] = a[y/x_1][x_1/x_2][x_2/y]$, where y is a variable that is fresh for all the involved items, namely $y \notin \{x_1, x_2\}$ and $y \# a$. Indeed, this is how one would define swapping from renaming on terms: using a fresh auxiliary variable y, and exploiting that such a fresh y exists and that its choice is immaterial for the end result. The next lemma shows that this style of definition also works abstractly, i.e., all it needs are the renset axioms plus Finite Support.

Lemma 5. The following hold for all $x_1, x_2 \in \mathsf{Var}$ and $a \in A$:

(1) There exists $y \in \mathsf{Var}$ such that $y \notin \{x_1, x_2\}$ and $y \# a$.
(2) For all $y, y' \in \mathsf{Var}$ such that $y \notin \{x_1, x_2\}$, $y \# a$, $y' \notin \{x_1, x_2\}$ and $y' \# a$, $a[y/x_1][x_1/x_2][x_2/y] = a[y'/x_1][x_1/x_2][x_2/y']$.

And one indeed obtains an operator satisfying the nominal axioms:

Proposition 6. If $(A, \_[\_/\_])$ is a renset satisfying Finite Support, then $(A, \_[\_ \wedge \_])$ is a nominal set. Moreover, $(A, \_[\_/\_])$ and $(A, \_[\_ \wedge \_])$ have the same notion of freshness, in that the freshness operator defined from renaming coincides with that defined from swapping.

The above construction is functorial, as I detail next. Given two nominal sets $\mathcal{A} = (A, \_[\_ \wedge \_])$ and $\mathcal{B} = (B, \_[\_ \wedge \_])$, a *nominal morphism* $f : \mathcal{A} \to \mathcal{B}$ is a function $f : A \to B$ with the property that it commutes with swapping, in that $(f\ a)[x \wedge y] = f(a[x \wedge y])$ for all $a \in A$ and $x, y \in \mathsf{Var}$. Nominal sets and nominal morphisms form a category that I will denote by $\underline{\mathit{Nom}}$. Similarly, let us define a morphism $f : \mathcal{A} \to \mathcal{B}$ between two rensets $\mathcal{A} = (A, \_[\_/\_])$ and $\mathcal{B} = (B, \_[\_])$ to be a function $f : A \to B$ that commutes with renaming, yielding the category $\underline{\mathit{Sbs}}$ of rensets. Let us write $\underline{\mathit{FSbs}}$ for the full subcategory of $\underline{\mathit{Sbs}}$ given by rensets that satisfy Finite Support. Let us define $F : \underline{\mathit{FSbs}} \to \underline{\mathit{Nom}}$ to be an operator on objects and morphisms that sends each finite-support renset to the above described nominal set constructed from it, and sends each substitutive morphism to itself.

Theorem 7. *F is a functor between $\underline{\mathit{FSbs}}$ and $\underline{\mathit{Nom}}$ which is injective on objects and full and faithful (i.e., bijective on morphisms).*

One may ask whether it is also possible to make the trip back: from nominal to rensets. The answer is negative, at least if one wants to retain the same notion of freshness, i.e., have the freshness predicate defined in the nominal set be identical to the one defined in the resulting renset. This is because swapping preserves the cardinality of the support, whereas renaming must be allowed to change it since it might perform a non-injective renaming. The following example captures this idea:

Counterexample. Let $\mathcal{A} = (A, \_[\_\wedge\_])$ be a nominal set such that all elements of A have their support consisting of exactly two variables, x and y (with $x \neq y$). (For example, A can be the set of all terms with these free variables—this is indeed a nominal subset of the term nominal set because it is closed under swapping.) Assume for a contradiction that $\_[\_/\_]$ is an operation on A that makes $(A, \_[\_/\_])$ a renset with its induced freshness operator equal to that of $\mathcal{A}$. Then, by the definition of A, $a[y/x]$ needs to have exactly two non-fresh variables. But this is impossible, since by Proposition 4(3), all the variables different from y (including x) must be fresh for $a[y/x]$. In particular, $\mathcal{A}$ is not in the image of the functor $F : \underline{\mathit{FSbs}} \to \underline{\mathit{Nom}}$, which is therefore not surjective on objects.

Thus, at an abstract algebraic level renaming can define swapping, but not the other way around. This is not too surprising, since swapping is fundamentally bijective whereas renaming is not; but it further validates our axioms for renaming, highlighting their ability to define a well-behaved swapping.

5 Recursion Based on Rensets

Proposition 3 shows that, in rensets, renaming can define freshness using only equality and universal or existential quantification over variables—without needing any cardinality condition like in the case of swapping. As I am about to discuss, this forms the basis of a characterization of terms as the initial algebra of an equational theory (Sect. 5.1) and an expressive recursion principle (Sect. 5.2) that fares better than the nominal one for interpretations in semantic domains (Sect. 5.3).

5.1 Equational Characterization of the Term Datatype

Rensets contain elements that are "term-like" in as much as there is a renaming operator on them satisfying familiar properties of renaming on terms. This similarity with terms can be strengthened by enriching rensets with operators having arities that match those of the term constructors.

A *constructor-enriched renset* (*CE renset* for short) is a tuple $\mathcal{A} = (A, \_[\_/\_], \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ where:

- $(A, \_[\_/\_])$ is a renset
- $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$ and $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$ are functions

such that the following hold for all $a, a_1, a_2 \in A$ and $x, y, z \in \mathsf{Var}$:

(S1) $(\mathsf{Vr}^{\mathcal{A}}\ x)[y/z] = \mathsf{Vr}^{\mathcal{A}}(x[y/z])$
(S2) $(\mathsf{Ap}^{\mathcal{A}}\ a_1\ a_2)[y/z] = \mathsf{Ap}^{\mathcal{A}}(a_1[y/z])\,(a_2[y/z])$
(S3) if $x \notin \{y, z\}$ then $(\mathsf{Lm}^{\mathcal{A}}\, x\ a)[y/z] = \mathsf{Lm}^{\mathcal{A}}\, x\,(a[y/z])$
(S4) $(\mathsf{Lm}^{\mathcal{A}}\, x\ a)[y/x] = \mathsf{Lm}^{\mathcal{A}}\, x\ a$
(S5) if $z \neq y$ then $\mathsf{Lm}^{\mathcal{A}}\, x\ (a[z/y]) = \mathsf{Lm}^{\mathcal{A}}\, y\ (a[z/y][y/x])$

Let us call $\mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}}$ the *constructors* of $\mathcal{A}$. (S1)–(S3) express the constructors' commutation with renaming (with capture-avoidance provisions in the case of (S3)), (S4) the lack of effect of substituting for a bound variable, and (S5) the possibility to rename a bound variable without changing the abstracted item (where the inner renaming of $z \neq y$ for y ensures the freshness of the "new name" y, hence its lack of interference with the other names in the "term-like" entity where the renaming takes place). All these are well-known to hold for terms:

Example. Terms with renaming and the constructors, namely $(\mathsf{Trm}, \_[\_/\_], \mathsf{Vr}, \mathsf{Ap}, \mathsf{Lm})$, form a CE renset which will be denoted by $\mathcal{Trm}$.

As it turns out, the CE renset axioms capture exactly the term structure $\mathcal{Trm}$, via initiality. The notion of *CE substitutive morphism* $f : \mathcal{A} \to \mathcal{B}$ between two CE rensets $\mathcal{A} = (A, \_[\_/\_], \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ and $\mathcal{B} = (B, \_[\_/\_], \mathsf{Vr}^{\mathcal{B}}, \mathsf{Ap}^{\mathcal{B}}, \mathsf{Lm}^{\mathcal{B}})$ is the expected one: a function $f : A \to B$ that is a substitutive morphism and also commutes with the constructors. Let us write $\underline{Sbs}_{\mathsf{CE}}$ for the category of CE rensets and morphisms.

Theorem 8. $\mathcal{Trm}$ is the initial CE renset, i.e., initial object in $\underline{Sbs}_{\mathsf{CE}}$.

Proof Idea. Let $\mathcal{A} = (A, \_[\_/\_], \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ be a CE renset. Instead of directly going after a function $f : \mathsf{Trm} \to A$, one first inductively defines a relation $R : \mathsf{Trm} \to A \to \mathsf{Bool}$, with inductive clauses reflecting the desired properties concerning the commutation with the constructors, e.g., $\frac{R\ t\ a}{R\ (\mathsf{Lm}\ x\ t)\ (\mathsf{Lm}^{\mathcal{A}}\ x\ a)}$. It suffices to prove that R is total and functional and preserves renaming, since that allows one to define a constructor- and renaming-preserving function (a morphism) f by taking $f\ t$ to be the unique a with $R\ t\ a$.

Proving that R is total is easy by standard induction on terms. Proving the other two properties, namely functionality and preservation of renaming, is more elaborate and requires their simultaneous proof together with a third property: that R preserves freshness. The simultaneous three-property proof follows by a form of "substitutive induction" on terms: Given a predicate $\phi : \mathsf{Trm} \to \mathsf{Bool}$, to show $\forall t \in \mathsf{Trm}.\ \phi\ t$ it suffices to show the following: (1) $\forall x \in \mathsf{Var}.\ \phi\ (\mathsf{Vr}\ x)$, (2) $\forall t_1, t_2 \in \mathsf{Trm}.\ \phi\ t_1 \,\&\, \phi\ t_2 \to \phi\,(\mathsf{Ap}\ t_1\ t_2)$, and (3) $\forall x \in \mathsf{Var}, t \in \mathsf{Trm}.\ (\forall s \in \mathsf{Trm}.\ \mathsf{Con}_{\_[\_/\_]}\ t\ s \to \phi\ s) \to \phi\,(\mathsf{Lm}\ x\ t)$, where $\mathsf{Con}_{\_[\_/\_]}\ t\ s$ means that t is connected to s by a chain of renamings.

Roughly speaking, R turns out to be functional because the λ-abstraction operator on the "term-like" inhabitants of A is, thanks to the axioms of CE renset, at least as non-injective as (i.e., identifies at least as many items as) the λ-abstraction operator on terms. □

Theorem 8 is the central result of this paper, from both practical and theoretical perspectives. Practically, it enables a useful form of recursion on terms (as I will discuss in the following sections). Theoretically, this is a characterization of terms as the initial algebra of an equational theory that only the most fundamental term operations, namely the constructors and renaming. The equational theory consists of the axioms of CE rensets (i.e., those of rensets plus (S1)–(S5)), which are an infinite set of unconditional equations—for example, axiom (S5) gives one equation for each pair of distinct variables y, z.

It is instructive to compare this characterization with the one offered by nominal logic, namely by Theorem 2. To do this, one first needs a lemma:

Lemma 9. Let $f : A \to B$ be a function between two nominal sets $\mathcal{A} = (A, \_[\_\wedge \_])$ and $\mathcal{B} = (B, \_[\_\wedge\_])$ and X a set of variables. Then f is supported by X if and only if $f(a[x \wedge y]) = (f\ a)[x \wedge y]$ for all $x, y \in \mathsf{Var} \smallsetminus X$.

Now Theorem 2 (with the variable avoidance set X taken to be $\emptyset$) can be rephrased as an initiality statement, as I describe below.

Let us define a *constructor-enriched nominal set* (*CE nominal set*) to be any tuple $\mathcal{A} = (A, \_[\_\wedge\_], \mathsf{Vr}^{\mathcal{A}}, \mathsf{Ap}^{\mathcal{A}}, \mathsf{Lm}^{\mathcal{A}})$ where $(A, \_[\_\wedge\_])$ is a nominal set and $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$, $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$ are operators on A such that the following properties hold for all $a, a_1, a_2 \in A$ and $x, y, z \in \mathsf{Var}$:

(N1) $(\mathsf{Vr}^{\mathcal{A}}\ x)[y \wedge z] = \mathsf{Vr}^{\mathcal{A}}(x[y \wedge z])$
(N2) $(\mathsf{Ap}^{\mathcal{A}}\ a_1\ a_2)[y \wedge z] = \mathsf{Ap}^{\mathcal{A}}(a_1[y \wedge z])\ (a_2[y \wedge z])$
(N3) $(\mathsf{Lm}^{\mathcal{A}}\ x\ a)[y \wedge z] = \mathsf{Lm}^{\mathcal{A}}\ (x[y \wedge z])\ (a[y \wedge z])$
(N4) $x \# \mathsf{Lm}\ x\ a$, i.e., $\{y \in \mathsf{Var} \mid (\mathsf{Lm}\ x\ a)[y \wedge x] \neq \mathsf{Lm}\ x\ a\}$ is finite.

The notion of *CE nominal morphism* is defined as the expected extension of that of nominal morphism: a function that commutes with swapping and the constructors. Let $\underline{Nom}_{\mathsf{CE}}$ be the category of CE nominal sets morphisms.

Theorem 10 ([39], rephrased)**.** $(\mathsf{Trm}, \_[\_ \wedge \_], \mathsf{Vr}, \mathsf{Ap}, \mathsf{Lm})$ is the initial CE nominal set, i.e., the initial object in $\underline{Nom}_{\mathsf{CE}}$.

The above theorem indeed corresponds exactly to Theorem 2 with $X = \emptyset$:

- the conditions (N1)–(N3) in the definition of CE nominal sets correspond (via Lemma 9) to the constructors being supported by $\emptyset$
- (N4) is the freshness condition for binders
- initiality, i.e., the existence of a unique morphism, is the same as the existence of the unique function $f : \mathsf{Trm} \to A$ stipulated in Theorem 2: commutation with the constructors is the Theorem 2 conditions (i)–(iii), and commutation with swapping means (via Lemma 9) f being supported by $\emptyset$.

Unlike the renaming-based characterization of terms (Theorem 8), the nominal logic characterization (Theorem 10) is not purely equational. This is due to a combination of two factors: (1) two of the axioms ((N4) and the Finite

Support condition) referring to freshness and (2) the impossibility of expressing freshness equationally from swapping. The problem seems fundamental, in that a nominal-style characterization does not seem to be expressible purely equationally. By contrast, while the freshness idea is implicit in the CE renset axioms, the freshness predicate itself is absent from Theorem 8.

5.2 Barendregt-Enhanced Recursion Principle

While Theorem 8 already gives a recursion principle, it is possible to improve it by incorporating Barendregt's variable convention (in the style of Theorem 2):

Theorem 11. Let X be a finite set, $(A, \_[\_/\_])$ a renset and $\mathsf{Vr}^{\mathcal{A}} : \mathsf{Var} \to A$, $\mathsf{Ap}^{\mathcal{A}} : A \to A \to A$ and $\mathsf{Lm}^{\mathcal{A}} : \mathsf{Var} \to A \to A$ some functions that satisfy the clauses (S1)–(S5) from the definition of CE renset, but only under the assumption that $x, y, z \notin X$. Then there exists a unique function $f : \mathsf{Trm} \to A$ such that th following hold:

(i) $f\,(\mathsf{Vr}\ x) = \mathsf{Vr}^{\mathcal{A}}\ x$ (ii) $f\,(\mathsf{Ap}\ t_1\ t_2) = \mathsf{Ap}^{\mathcal{A}}\,(f\ t_1)\,(f\ t_2)$
(iii) $f\,(\mathsf{Lm}\ x\ t) = \mathsf{Lm}^{\mathcal{A}}\ x\ (f\ t)$ if $x \notin X$ (iv) $f\,(t[y/z]) = (f\ t)[y/z]$ if $y, z \notin X$

Proof Idea. The constructions in the proof of Theorem 8 can be adapted to avoid clashing with the finite set of variables X. For example, the clause for λ-abstraction in the inductive definition of the relation R becomes $\frac{x \notin X \qquad R\ t\ a}{R\ (\mathsf{Lm}\ x\ t)\ (\mathsf{Lm}^{\mathcal{A}}\ x\ a)}$ and preservation of renaming and freshness are also formulated to avoid X. Totality is still ensured thanks to the possibility of renaming bound variables—in terms and inhabitants of A alike (via the modified axiom (S5)). □

The above theorem says that if the structure $\mathcal{A}$ is assumed to be "almost" a CE set, save for additional restrictions involving the avoidance of X, then there exists a unique "almost"-morphism—satisfying the CE substitutive morphism conditions restricted so that the bound and renaming-participating variables avoid X. It is the renaming-based counterpart of the nominal Theorem 2.

In regards to the relative expressiveness of these two recursion principles (Theorems 11 and 2), it seems difficult to find an example that is definable by one but not by the other. In particular, my principle can seamlessly define standard nominal examples [39,40] such as the length of a term, the counting of λ-abstractions or of the free-variables occurrences, and term-for-variable substitution—[45, Appendix A] gives details. However, as I am about to discuss, I found an important class of examples where my renaming-based principle is significantly easier to deploy: that of interpreting syntax in semantic domains.

5.3 Extended Example: Semantic Interpretation

Semantic interpretations, also known as denotations (or denotational semantics), are pervasive in the meta-theory of logics and λ-calculi, for example when interpretating first-order logic (FOL) formulas in FOL models, or untyped or simply-typed λ-calculus or higher-order logic terms in specific models (such as full-frame or Henkin models). In what follows, I will focus on λ-terms and Henkin models, but the ideas discussed apply broadly to any kind of statically scoped interpretation of terms or formulas involving binders.

Let D be a set and $\mathsf{ap} : D \to D \to D$ and $\mathsf{lm} : (D \to D) \to D$ be operators modeling semantic notions of application and abstraction. An environment will be a function $\xi : \mathsf{Var} \to D$. Given $x, y \in \mathsf{Var}$ and $d, e \in D$, let us write $\xi\langle x := d\rangle$ for ξ updated with value d for x (i.e., acting like ξ on all variables except for x where it returns d); and let us write $\xi\langle x := d, y := e\rangle$ instead of $\xi\langle x := d\rangle\langle y := e\rangle$.

Say one wants to interpret terms in the semantic domain D in the context of environments, i.e., define the function $\mathsf{sem} : \mathsf{Trm} \to (\mathsf{Var} \to D) \to D$ that maps syntactic to semantic constructs; e.g., one would like to have:

- $\mathsf{sem}\,(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ x))\ \xi = \mathsf{lm}(d \mapsto \mathsf{ap}\ d\ d)$ (regardless of ξ)
- $\mathsf{sem}\,(\mathsf{Lm}\ x\ (\mathsf{Ap}\ x\ y))\ \xi = \mathsf{lm}(d \mapsto \mathsf{ap}\ d\ (\xi\ y))$ (assuming $x \neq y$)

where I use $d \mapsto \ldots$ to describe functions in $D \to D$, e.g., $d \mapsto \mathsf{ap}\ d\ d$ is the function sending every $d \in D$ to $\mathsf{ap}\ d\ d$.

The definition should therefore naturally go recursively by the clauses:
(1) $\mathsf{sem}\,(\mathsf{Vr}\ x)\,\xi = \xi\ x$ (2) $\mathsf{sem}\,(\mathsf{Ap}\ t_1\,t_2)\,\xi = \mathsf{ap}\,(\mathsf{sem}\ t_1\,\xi)\,(\mathsf{sem}\ t_2\,\xi)$
(3) $\mathsf{sem}\,(\mathsf{Lm}\ x\ t)\,\xi = \mathsf{lm}\,(d \mapsto \mathsf{sem}\ t\,(\xi\langle x := d\rangle))$

Of course, since Trm is not a free datatype, these clauses do not work out of the box, i.e., do not form a definition (yet)—this is where binding-aware recursion principles such as Theorems 11 and 2 could step in. I will next try them both.

The three clauses above already determine constructor operations $\mathsf{Vr}^{\mathcal{I}}$, $\mathsf{Ap}^{\mathcal{I}}$ and $\mathsf{Lm}^{\mathcal{I}}$ on the set of interpretations, $I = (\mathsf{Var} \to D) \to D$, namely:

- $\mathsf{Vr}^{\mathcal{I}} : \mathsf{Var} \to I$ by $\mathsf{Vr}^{\mathcal{I}}\ x\ i\ \xi = \xi\ x$
- $\mathsf{Ap}^{\mathcal{I}} : I \to I \to I$ by $\mathsf{Ap}^{\mathcal{I}}\ i_1\,i_2\ \xi = \mathsf{ap}\,(i_1\,\xi)\,(i_2\,\xi)$
- $\mathsf{Lm}^{\mathcal{I}} : \mathsf{Var} \to I \to I$ by $\mathsf{Lm}^{\mathcal{I}}\ x\ i\ \xi = \mathsf{lm}\,(d \mapsto i\,(\xi\langle x := d\rangle))$

To apply the renaming-based recursion principle from Theorem 11, one must further define a renaming operator on I. Since the only chance to successfully apply this principle is if sem commutes with renaming, the definition should be inspired by the question: How can $\mathsf{sem}(t[y/x])$ be determined from $\mathsf{sem}\ t$, y and x? The answer is (4) $\mathsf{sem}\,(t[y/x])\,\xi = (\mathsf{sem}\ t)\ (\xi\langle x := \xi\ y\rangle)$, yielding an operator $[\_/\_]^{\mathcal{I}} : I \to \mathsf{Var} \to \mathsf{Var} \to I$ defined by $i\,[y/x]^{\mathcal{I}}\,\xi = i\,(\xi\langle x := \xi\ y\rangle)$.

It is not difficult to verify that $\mathcal{I} = (I, [\_/\_]^{\mathcal{I}}, \mathsf{Vr}^{\mathcal{I}}, \mathsf{Ap}^{\mathcal{I}}, \mathsf{Lm}^{\mathcal{I}})$ is a CE renset—for example, Isabelle's automatic methods discharge all the goals. This means Theorem 11 (or, since here one doesn't need Barendregt's variable convention, already Theorem 8) is applicable, and gives us a unique function sem that commutes with the constructors, i.e., satisfies clauses (1)–(3) (which are instances of the clauses (i)–(iii) from Theorem 11), and additionally commutes with renaming, i.e., satisfies clause (4) (which is an instances of the clause (iv) from Theorem 11).

On the other hand, to apply nominal recursion for defining sem, one must identify a swapping operator on I. Similarly to the case of renaming, this identification process is guided by the goal of determining $\mathsf{sem}(t[x{\wedge}y])$ from $\mathsf{sem}\ t$, x and

y, leading to (4') $\mathsf{sem}\,(t[x \wedge y])\,\xi = \mathsf{sem}\; t\;(\xi\langle x := \xi\; y, y := \xi\; x\rangle)$, which yields the definition of $[\_\wedge\_]^{\mathcal{I}}$ by $i\,[x \wedge y]^{\mathcal{I}}\,\xi = i\,(\xi\langle x := \xi\; y, y := \xi\; x\rangle)$. However, as pointed out by Pitts [39, §6.3] (in the slightly different context of interpreting simply-typed λ-calculus), the nominal recursor (Theorem 2) does *not* directly apply (hence neither does my reformulation based on CE nominal sets, Theorem 10). This is because, in my terminology, the structure $\mathcal{I} = (I, [\_\wedge\_]^{\mathcal{I}}, \mathsf{Vr}^{\mathcal{I}}, \mathsf{Ap}^{\mathcal{I}}, \mathsf{Lm}^{\mathcal{I}})$ is not a CE nominal set. The problematic condition is FCB (the freshness condition for binders), requiring that $x \,\#^{\mathcal{I}}\, (\mathsf{Lm}^{\mathcal{I}}\; x\; i)$ holds for all $i \in I$. Expanding the definition of $\#^{\mathcal{I}}$ (the nominal definition of freshness from swapping, recalled in Sect. 2.2) and the definitions of $[\_\wedge\_]^{\mathcal{I}}$ and $\mathsf{Lm}^{\mathcal{I}}$, one can see that $x \,\#^{\mathcal{I}}\, (\mathsf{Lm}^{\mathcal{I}}\; x\; i)$ means the following:
$\mathsf{Im}\;(d \mapsto i\,(\xi\langle x := \xi\, y, y := \xi\, x\rangle\langle x := d\rangle)) = \mathsf{Im}\;(d \mapsto i\,(\xi\langle x := d\rangle))$, i.e., $\mathsf{Im}\;(d \mapsto i\,(\xi\langle x := d, y := \xi\, x\rangle) = \mathsf{Im}\;(d \mapsto i\,(\xi\langle x := d\rangle))$, holds for all but a finite number of variables y.

The only chance for the above to be true is if i, when applied to an environment, ignores the value of y in that environment for all but a finite number of variables y; in other words, i only analyzes the value of a finite number of variables in that environment—but this is not guaranteed to hold for arbitrary elements $i \in I$. To repair this, Pitts engages in a form of induction-recursion [17], carving out from I a smaller domain that is still large enough to interpret all terms, then proving that both FCB and the other axioms hold for this restricted domain. It all works out in the end, but the technicalities are quite involved.

Although FCB is not required by the renaming-based principle, note incidentally that this condition would actually be true (and immediate to check) if working with freshness defined not from swapping but from renaming. Indeed, the renaming-based version of $x \;\#^{\mathcal{I}}\, (\mathsf{Lm}^{\mathcal{I}}\; x\;\; i)$ says that $\mathsf{Im}\;(d \mapsto i\,(\xi\langle x := \xi\, y\rangle\langle x := d\rangle)) = \mathsf{Im}\;(d \mapsto i\,(\xi\langle x := d\rangle))$ holds for all y (or at least for some $y \neq x$)—which is immediate since $\xi\langle x := \xi\, y\rangle\langle x := d\rangle = \xi\langle x := d\rangle$. This further illustrates the idea that semantic domains 'favor' renaming over swapping.

In conclusion, for interpreting syntax in semantic domains, my renaming-based recursor is trivial to apply, whereas the nominal recursor requires some fairly involved additional definitions and proofs.

6 Conclusion and Related Work

This paper introduced and studied rensets, contributing (1) theoretically, a minimalistic equational characterization of the datatype of terms with bindings and (2) practically, an addition to the formal arsenal for manipulating syntax with bindings. It is part of a longstanding line of work by myself and collaborators on exploring convenient definition and reasoning principles for bindings [25,27,43,46,47], and will be incorporated into the ongoing implementation of a new Isabelle definitional package for binding-aware datatypes [12].

Initial Model Characterizations of the Terms Datatype. My results provide a truly elementary characterization of terms with bindings, as an "ordinary" datatype specified by the fundamental operations only (the constructors plus

	Fiore et al. [22] Hofmann [29]	Pitts [39]	Urban et al. [57,56]	Norrish [33]	Popescu &Gunter [46]	Gheri& Popescu [25]	This paper
Paradigm	nameless	nameful	nameful	nameful	nameful	nameful	nameful
Barendregt?	n/a	yes	yes	yes	no	no	yes
Underlying category	$Set^{\mathbb{F}}$	*Set*	*Set*	*Set*	*Set*	*Set*	*Set*
Required operations/ relations	ctors, rename, free-vars	ctors, perm	ctors, perm	ctors, swap, free-vars	ctors, term/var subst, fresh	ctors, swap, fresh	ctors, rename
Required properties	functoriality, naturality	Horn clauses, fresh-def, fin-supp	Horn clauses, fresh-def	Horn clauses	Horn clauses	Horn clauses	equations

Fig. 1. Initial model characterizations of the datatype of terms with bindings "ctors" = "constructors", "perm" = "permutation", "fresh" = "the freshness predicate", "fresh-def" = "clause for defining the freshness predicate", "fin-supp" = "Finite Support"

variable-for-variable renaming) and some equations (those defining CE rensets). As far as specification simplicity goes, this is "the next best thing" after a completely free datatype such as those of natural numbers or lists.

Figure 1 shows previous characterizations from the literature, in which terms with bindings are identified as an initial model (or algebra) of some kind. For each of these, I indicate (1) the employed reasoning paradigm, (2) whether the initiality/recursion theorem features an extension with Barendregt's variable convention, (3) the underlying category (from where the carriers of the models are taken), (4) the operations and relations on terms to which the models must provide counterparts and (5) the properties required on the models.

While some of these results enjoy elegant mathematical properties of intrinsic value, my main interest is in the recursors they enable, specifically in the ease of deploying these recursors. That is, I am interested in how easy it is in principle to organize the target domain as a model of the requested type, hence obtain the desired morphism, i.e., get the recursive definition done. By this measure, elementary approaches relying on standard FOL-like models whose carriers are sets rather than pre-sheaves have an advantage. Also, it seems intuitive that a recursor is easier to apply if there are fewer operators, and fewer and structurally simpler properties required on its models—although empirical evidence of successfully deploying the recursor in practice should complement the simplicity assessment, to ensure that simplicity is not sponsored by lack of expressiveness.

The first column in Fig. 1's table contains an influential representative of the nameless paradigm: the result obtained independently by Fiore et al. [22] and Hofmann [29] characterizing terms as initial in the category of algebras over the pre-sheaf topos $\mathit{Set}^{\mathbb{F}}$, where $\mathbb{F}$ is the category of finite ordinals and functions between them. The operators required by algebras are the constructors, as well as the free-variable operator (implicitly as part of the separation on levels) and the injective renamings (as part of the functorial structure). The algebra's carrier is required to be a functor and the constructors to be natural transformations. There are several variations of this approach, e.g., [5,11,29], some implemented in proof assistants, e.g., [3,4,31].

The other columns refer to initiality results that are more closely related to mine. They take place within the nameful paradigm, and they all rely on elementary models (with set carriers). Pitts's already discussed nominal recursor [39] (based on previous work by Gabbay and Pitts [23]) employs the constructors and permutation (or swapping), and requires that its models satisfy some Horn clauses for constructors, permutation and freshness, together with the second-order properties that (1) define freshness from swapping and (2) express Finite Support. Urban et al.'s version [56,57] implemented in Isabelle/Nominal is an improvement of Pitts's in that it removes the Finite Support requirement from the models—which is practically significant because it enables non-finitely supported target domains for recursion. Norrish's result [33] is explicitly inspired by nominal logic, but renounces the definability of the free-variable operator from swapping—with the price of taking both swapping and free-variables as primitives. My previous work with Gunter and Gheri takes as primitives either term-for-variable substitution and freshness [46] or swapping and freshness [25], and requires properties expressed by different Horn clauses (and does not explore a Barendregt dimension, like Pitts, Urban et al. and Norrish do). My previous focus on term-for-variable substitution [46] (as opposed to renaming, i.e., variable-for-variable substitution) impairs expressiveness—for example, the depth of a term is not definable using a recursor based on term-for-variable substitution because we cannot say how term-for-variable substitution affects the depth of a term based on its depth and that of the substitutee alone. My current result based on rensets keeps freshness out of the primitive operators base (like nominal logic does), and provides an unconditionally equational characterization using only constructors and renaming. The key to achieving this minimality is the simple expression of freshness from renaming in my axiomatization of rensets. In future work, I plan a systematic formal comparison of the relative expressiveness of all these nameful recursors.

Recursors in Other Paradigms. Figure 1 focuses on nameful recursors, while only the Fiore et al./Hofmann recursor for the sake of a rough comparison with the nameless approach. I should stress that such a comparison is necessarily rough, since the nameless recursors do not give the same "payload" as the nameful ones. This is because of the handling of bound variables. In the nameless paradigm, the λ-constructor does not explicitly take a variable as an input, as in $\mathsf{Lm}\ x\ t$, i.e., does not have type $\mathsf{Var} \to \mathsf{Trm} \to \mathsf{Trm}$. Instead, the bindings are indicated through nameless pointers to positions in a term. So the nameless λ-constructor, let's call it NLm, takes only a term, as in $\mathsf{NLm}\ t$, i.e., has type $\mathsf{Trm} \to \mathsf{Trm}$ or a scope-safe (polymorphic or dependently-typed) variation of this, e.g., $\prod_{n\in\mathbb{F}} \mathsf{Trm}_n \to \mathsf{Trm}_{n+1}$ [22,29] or $\prod_{\alpha\in\mathsf{Type}} \mathsf{Trm}_\alpha \to \mathsf{Trm}_{\alpha+\mathsf{unit}}$ [5,11]. The λ-constructor is of course matched by operators in the considered models, which appears in the clauses of the functions f defined recursively on terms: Instead of a clause of the form $f\ (\mathsf{Lm}\ x\ t) = \langle$expression depending on x and $f\,t\rangle$ from the nameful paradigm, in the nameless paradigm one gets a clause of the form $f\ (\mathsf{NLm}\ t) = \langle$expression depending on $f\,t\rangle$. A nameless recursor is usually easier to prove correct and easier to apply because the nameless constructor NLm is free—whereas a nameful recursor must wrestle with the non-freeness of Lm, handled by verifying certain properties of the target models. However, once the definition is done, having nameful clauses pays off by allowing "textbook-style" proofs that stay close to the informal presentation of a calculus or logic, whereas with the nameless definition some additional index shifting bureaucracy is necessary. (See [9] for a detailed discussion, and [14] for a hybrid solution.)

A comparison of nameful recursion with HOAS recursion is also generally difficult, since major HOAS frameworks such as Abella [7], Beluga [37] or Twelf [36] are developed within non-standard logical foundations, allowing a λ-constructor of type $(\mathsf{Trm} \to \mathsf{Trm}) \to \mathsf{Trm}$, which is not amenable to typical well-foundedness based recursion but requires some custom solutions (e.g., [21,50]). However, the *weak HOAS* variant [16,27] employs a constructor of the form $\mathsf{WHLm} : (\mathsf{Var} \to \mathsf{Trm}) \to \mathsf{Trm}$ which *is* recursable, and in fact yields a free datatype, let us call it WHTrm—one generated by $\mathsf{WHVr} : \mathsf{Var} \to \mathsf{WHTrm}$, $\mathsf{WHAp} : \mathsf{WHTrm} \to \mathsf{WHTrm} \to \mathsf{WHTrm}$ and WHLm. WHTrm contains (natural encodings of) all terms but also additional entities referred to as "exotic terms". Partly because of the exotic terms, this free datatype by itself is not very helpful for recursively defining useful functions on terms. But the situation is dramatically improved if one employs a variant of weak HOAS called *parametric HOAS (PHOAS)* [15], i.e., takes Var not as a fixed type but as a type parameter (type variable) and works with $\prod_{\mathsf{Var}\in\mathsf{Type}} \mathsf{Trm}_{\mathsf{Var}}$; this enables many useful definitions by choosing a suitable type Var (usually large enough to make the necessary distinctions) and then performing standard recursion. The functions definable in the style of PHOAS seem to be exactly those definable via the semantic domain interpretation pattern (Sect. 5.3): Choosing the instantiation of Var to a type T corresponds to employing environments in $\mathsf{Var} \to T$. (I illustrate this at the end of [45, Appendix A] by showing the semantic-domain version of a PHOAS example.)

As a hybrid nameful/HOAS approach we can count Gordon and Melham's characterization of the datatype of terms [26], which employs the nameful constructors but formulates recursion treating Lm as if recursing in the weak-HOAS datatype WHTrm. Norrish's recursor [33] (a participant in Fig. 1) has been inferred from Gordon and Melham's one. Weak-HOAS recursion also has interesting connections with nameless recursion: In presheaf toposes such as those employed by Fiore et al. [22], Hofmann [29] and Ambler et al. [6], for any object T the function space $\mathsf{Var} \Rightarrow T$ is isomorphic to the De Bruijn level shifting transformation applied to T; this effectively equates the weak-HOAS and nameless recursors. A final cross-paradigm note: In themselves, nominal sets are not confined to the nameful paradigm; their category is equivalent [23] to the Schanuel topos [30], which is attractive for pursuing the nameless approach.

Axiomatizations of Renaming. In his study of name-passing process calculi, Staton [52] considers an enrichment of nominal sets with renaming (in addition to swapping) and axiomatizes renaming with the help of the nominal (swapping-defined) freshness predicate. He shows that the resulted category is equivalent to the non-injective renaming counterpart of the Schanuel topos (i.e., the subcategory of $Set^{\mathbb{F}}$ consisting of functors that preserve pullbacks of monos). Gabbay and Hofmann [24] provide an elementary characterization of the above category, in terms of *nominal renaming sets*, which are sets equipped with a multiple-variable-renaming action satisfying identity and composition laws, and a form of Finite Support (FS). Nominal renaming sets seem very related to rensets satisfying FS. Indeed, any nominal renaming set forms a FS-satisfying renset when restricted to single-variable renaming. Conversely, I conjecture that any FS-satisfying renset gives rise to a nominal renaming set. This correspondence seems similar to the one between the permutation-based and swapping-based alternative axiomatizations of nominal sets—in that the two express the same concept up to an isomorphism of categories. In their paper, Gabbay and Hofmann do not study renaming-based recursion, beyond noting the availability of a recursor stemming from the functor-category view (which, as I discussed above, enables nameless recursion with a weak-HOAS flavor). Pitts [41] introduces *nominal sets with* 01-*substitution structure*, which axiomatize substitution of one of two possible constants for variables on top of the nominal axiomatization, and proves that they form a category that is equivalent with that of cubical sets [10], hence relevant for the univalent foundations [54].

Other Work. Sun [53] develops universal algebra for first-order languages with bindings (generalizing work by Aczel [2]) and proves a completeness theorem. In joint work with Roşu [48], I develop first-order logic and prove completeness on top of a generic syntax with axiomatized free-variables and substitution.

Renaming Versus Swapping and Nominal Logic, Final Round. I believe that my work complements rather than competes with nominal logic. My results do not challenge the swapping-based approach to defining syntax (defining the alpha-equivalence on pre-terms and quotienting to obtain terms) recommended by nominal logic, which is more elegant than a renaming-based alternative; but my easier-to-apply recursor can be a useful addition even on top of the nominal substratum. Moreover, some of my constructions are explicitly inspired by the nominal ones. For example, I started by adapting the nominal idea of defining freshness from swapping before noticing that renaming enables a simpler formulation. My formal treatment of Barendregt's variable convention also originates from nominal logic—as it turns out, this idea works equally well in my setting. In fact, I came to believe that the possibility of a Barendregt enhancement is largely orthogonal to the particularities of a binding-aware recursor. In future work, I plan to investigate this, i.e., seek general conditions under which an initiality principle (such as Theorems 10 and 8) is amenable to a Barendregt enhancement (such as Theorems 2 and 11, respectively).

Acknowledgments. I am grateful to the IJCAR reviewers for their insightful comments and suggestions, and for pointing out related work.

References

1. Abel, A., et al.: Poplmark reloaded: mechanizing proofs by logical relations. J. Funct. Program. **29**, e19 (2019). https://doi.org/10.1017/S0956796819000170
2. Aczel, P.: Frege structures and notations in propositions, truth and set. In: The Kleene Symposium, pp. 31–59. North Holland (1980)
3. Allais, G., Atkey, R., Chapman, J., McBride, C., McKinna, J.: A type and scope safe universe of syntaxes with binding: their semantics and proofs. Proc. ACM Program. Lang. **2**(International Conference on Functional Programming (ICFP)), 90:1–90:30 (2018). https://doi.acm.org/10.1145/3236785
4. Allais, G., Chapman, J., McBride, C., McKinna, J.: Type-and-scope safe programs and their proofs. In: Bertot, Y., Vafeiadis, V. (eds.) Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, 16–17 January 2017, pp. 195–207. ACM (2017). https://doi.org/10.1145/3018610.3018613
5. Altenkirch, T., Reus, B.: Monadic presentations of lambda terms using generalized inductive types. In: Flum, J., Rodriguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 453–468. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48168-0_32
6. Ambler, S.J., Crole, R.L., Momigliano, A.: A definitional approach to primitivexs recursion over higher order abstract syntax. In: Eighth ACM SIGPLAN International Conference on Functional Programming, Workshop on Mechanized Reasoning About Languages with Variable Binding, MERLIN 2003, Uppsala, Sweden, August 2003. ACM (2003). https://doi.org/10.1145/976571.976572
7. Baelde, D., et al.: Abella: a system for reasoning about relational specifications. J. Formaliz. Reason. **7**(2), 1–89 (2014). https://doi.org/10.6092/issn.1972-5787/4650
8. Barendregt, H.P.: The Lambda Calculus: Its Syntax and Semantics, Studies in Logic, vol. 40. Elsevier (1984)
9. Berghofer, S., Urban, C.: A head-to-head comparison of de Bruijn indices and names. Electr. Notes Theor. Comput. Sci. **174**(5), 53–67 (2007). https://doi.org/10.1016/j.entcs.2007.01.018
10. Bezem, M., Coquand, T., Huber, S.: A model of type theory in cubical sets. In: Matthes, R., Schubert, A. (eds.) 19th International Conference on Types for Proofs and Programs, TYPES 2013, 22–26 April 2013, Toulouse, France. LIPIcs, vol. 26, pp. 107–128. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013). https://doi.org/10.4230/LIPIcs.TYPES.2013.107
11. Bird, R.S., Paterson, R.: De Bruijn notation as a nested datatype. J. Funct. Program. **9**(1), 77–91 (1999). https://doi.org/10.1017/S0956796899003366
12. Blanchette, J.C., Gheri, L., Popescu, A., Traytel, D.: Bindings as bounded natural functors. Proc. ACM Program. Lang. **3**(POPL), 22:1–22:34 (2019). https://doi.org/10.1145/3290335
13. de Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. Indag. Math. **75**(5), 381–392 (1972). https://doi.org/10.1016/1385-7258(72)90034-0
14. Charguéraud, A.: The locally nameless representation. J. Autom. Reason. **49**(3), 363–408 (2012). https://doi.org/10.1007/s10817-011-9225-2

15. Chlipala, A.: Parametric higher-order abstract syntax for mechanized semantics. In: Hook, J., Thiemann, P. (eds.) International Conference on Functional Programming (ICFP) 2008, pp. 143–156. ACM (2008). https://doi.org/10.1145/1411204.1411226
16. Despeyroux, J., Felty, A., Hirschowitz, A.: Higher-order abstract syntax in Coq. In: Dezani-Ciancaglini, M., Plotkin, G. (eds.) TLCA 1995. LNCS, vol. 902, pp. 124–138. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0014049
17. Dybjer, P.: A general formulation of simultaneous inductive-recursive definitions in type theory. J. Symb. Log. **65**(2), 525–549 (2000). https://doi.org/10.2307/2586554
18. Felty, A.P., Momigliano, A.: Hybrid: a definitional two-level approach to reasoning with higher-order abstract syntax. J. Autom. Reason. **48**(1), 43–105 (2012). https://doi.org/10.1007/s10817-010-9194-x
19. Felty, A.P., Momigliano, A., Pientka, B.: The next 700 challenge problems for reasoning with higher-order abstract syntax representations - part 2 - a survey. J. Autom. Reason. **55**(4), 307–372 (2015). https://doi.org/10.1007/s10817-015-9327-3
20. Felty, A.P., Momigliano, A., Pientka, B.: An open challenge problem repository for systems supporting binders. In: Cervesato, I., Chaudhuri, K. (eds.) Proceedings Tenth International Workshop on Logical Frameworks and Meta Languages: Theory and Practice, LFMTP 2015, Berlin, Germany, 1 August 2015. EPTCS, vol. 185, pp. 18–32 (2015). https://doi.org/10.4204/EPTCS.185.2
21. Ferreira, F., Pientka, B.: Programs using syntax with first-class binders. In: Yang, H. (ed.) ESOP 2017. LNCS, vol. 10201, pp. 504–529. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54434-1_19
22. Fiore, M.P., Plotkin, G.D., Turi, D.: Abstract syntax and variable binding. In: Logic in Computer Science (LICS) 1999, pp. 193–202. IEEE Computer Society (1999). https://doi.org/10.1109/LICS.1999.782615
23. Gabbay, M., Pitts, A.M.: A new approach to abstract syntax involving binders. In: Logic in Computer Science (LICS) 1999, pp. 214–224. IEEE Computer Society (1999). https://doi.org/10.1109/LICS.1999.782617
24. Gabbay, M.J., Hofmann, M.: Nominal renaming sets. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 158–173. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89439-1_11
25. Gheri, L., Popescu, A.: A formalized general theory of syntax with bindings: extended version. J. Autom. Reason. **64**(4), 641–675 (2020). https://doi.org/10.1007/s10817-019-09522-2
26. Gordon, A.D., Melham, T.: Five axioms of alpha-conversion. In: Goos, G., Hartmanis, J., van Leeuwen, J., von Wright, J., Grundy, J., Harrison, J. (eds.) TPHOLs 1996. LNCS, vol. 1125, pp. 173–190. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0105404
27. Gunter, E.L., Osborn, C.J., Popescu, A.: Theory support for weak higher order abstract syntax in Isabelle/HOL. In: Cheney, J., Felty, A.P. (eds.) Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP) 2009, pp. 12–20. ACM (2009). https://doi.org/10.1145/1577824.1577827
28. Harper, R., Honsell, F., Plotkin, G.D.: A framework for defining logics. In: Logic in Computer Science (LICS) 1987, pp. 194–204. IEEE Computer Society (1987). https://doi.org/10.1145/138027.138060
29. Hofmann, M.: Semantical analysis of higher-order abstract syntax. In: Logic in Computer Science (LICS) 1999, pp. 204–213. IEEE Computer Society (1999). https://doi.org/10.1109/LICS.1999.782616

30. Johnstone, P.T.: Quotients of decidable objects in a topos. Math. Proc. Camb. Philos. Soc. **93**, 409–419 (1983). https://doi.org/10.1017/S0305004100060734
31. Kaiser, J., Schäfer, S., Stark, K.: Binder aware recursion over well-scoped de bruijn syntax. In: Andronick, J., Felty, A.P. (eds.) Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, 8–9 January 2018, pp. 293–306. ACM (2018). https://doi.org/10.1145/3167098
32. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL—A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9
33. Norrish, M.: Recursive function definition for types with binders. In: Slind, K., Bunker, A., Gopalakrishnan, G. (eds.) TPHOLs 2004. LNCS, vol. 3223, pp. 241–256. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30142-4_18
34. Paulson, L.C.: The foundation of a generic theorem prover. J. Autom. Reason. **5**(3), 363–397 (1989). https://doi.org/10.1007/BF00248324
35. Pfenning, F., Elliott, C.: Higher-order abstract syntax. In: Wexelblat, R.L. (ed.) Programming Language Design and Implementation (PLDI) 1988, pp. 199–208. ACM (1988). https://doi.org/10.1145/53990.54010
36. Pfenning, F., Schürmann, C.: System description: twelf — a meta-logical framework for deductive systems. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 202–206. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48660-7_14
37. Pientka, B.: Beluga: programming with dependent types, contextual data, and contexts. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) FLOPS 2010. LNCS, vol. 6009, pp. 1–12. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12251-4_1
38. Pitts, A.M.: Nominal logic, a first order theory of names and binding. Inf. Comput. **186**(2), 165–193 (2003). https://doi.org/10.1016/S0890-5401(03)00138-X
39. Pitts, A.M.: Alpha-structural recursion and induction. J. ACM **53**(3), 459–506 (2006). https://doi.org/10.1145/1147954.1147961
40. Pitts, A.M.: Nominal Sets: Names and Symmetry in Computer Science. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge (2013)
41. Pitts, A.M.: Nominal presentation of cubical sets models of type theory. In: Herbelin, H., Letouzey, P., Sozeau, M. (eds.) 20th International Conference on Types for Proofs and Programs (TYPES 2014). Leibniz International Proceedings in Informatics (LIPIcs), vol. 39, pp. 202–220. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2015). http://drops.dagstuhl.de/opus/volltexte/2015/5498
42. Pollack, R., Sato, M., Ricciotti, W.: A canonical locally named representation of binding. J. Autom. Reason. **49**(2), 185–207 (2012). https://doi.org/10.1007/s10817-011-9229-y
43. Popescu, A.: Contributions to the theory of syntax with bindings and to process algebra. Ph.D. thesis, University of Illinois at Urbana-Champaign (2010). https://www.andreipopescu.uk/pdf/thesisUIUC.pdf
44. Popescu, A.: Renaming-Enriched Sets. Arch. Formal Proofs 2022 (2022). https://www.isa-afp.org/entries/Renaming_Enriched_Sets.html
45. Popescu, A.: Rensets and renaming-based recursion for syntax with bindings. arXiv (2022). https://arxiv.org/abs/2205.09233

46. Popescu, A., Gunter, E.L.: Recursion principles for syntax with bindings and substitution. In: Chakravarty, M.M.T., Hu, Z., Danvy, O. (eds.) Proceeding of the 16th ACM SIGPLAN International Conference on Functional Programming, ICFP 2011, Tokyo, Japan, 19–21 September 2011, pp. 346–358. ACM (2011). https://doi.org/10.1145/2034773.2034819
47. Popescu, A., Gunter, E.L., Osborn, C.J.: Strong normalization for system F by HOAS on top of FOAS. In: Logic in Computer Science (LICS) 2010, pp. 31–40. IEEE Computer Society (2010). https://doi.org/10.1109/LICS.2010.48
48. Popescu, A., Roşu, G.: Term-generic logic. Theor. Comput. Sci. **577**, 1–24 (2015)
49. Schäfer, S., Tebbi, T., Smolka, G.: Autosubst: reasoning with de Bruijn terms and parallel substitutions. In: Urban, C., Zhang, X. (eds.) ITP 2015. LNCS, vol. 9236, pp. 359–374. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22102-1_24
50. Schürmann, C., Despeyroux, J., Pfenning, F.: Primitive recursion for higher-order abstract syntax. Theor. Comput. Sci. **266**(1–2), 1–57 (2001). https://doi.org/10.1016/S0304-3975(00)00418-7
51. Stark, K.: Mechanising syntax with binders in Coq. Ph.D. thesis, Saarland University, Saarbrücken, Germany (2020). https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/28822
52. Staton, S.: Name-passing process calculi: operational models and structural operational semantics. Technical report, UCAM-CL-TR-688, University of Cambridge, Computer Laboratory (2007). https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-688.pdf
53. Sun, Y.: An algebraic generalization of Frege structures–binding algebras. Theor. Comput. Sci. **211**(1–2), 189–232 (1999)
54. The Univalent Foundations Program: Homotopy Type Theory. Univalent Foundations of Mathematics. Institute for Advanced Study (2013). https://homotopytypetheory.org/book
55. Traytel, D., Popescu, A., Blanchette, J.C.: Foundational, compositional (co)datatypes for higher-order logic: category theory applied to theorem proving. In: Logic in Computer Science (LICS) 2012, pp. 596–605. IEEE Computer Society (2012). https://doi.org/10.1109/LICS.2012.75
56. Urban, C.: Nominal techniques in Isabelle/HOL. J. Autom. Reason. **40**(4), 327–356 (2008). https://doi.org/10.1007/s10817-008-9097-2
57. Urban, C., Berghofer, S.: A recursion combinator for nominal datatypes implemented in Isabelle/HOL. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 498–512. Springer, Heidelberg (2006). https://doi.org/10.1007/11814771_41
58. Urban, C., Berghofer, S., Norrish, M.: Barendregt's variable convention in rule inductions. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 35–50. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73595-3_4
59. Urban, C., Kaliszyk, C.: General bindings and alpha-equivalence in Nominal Isabelle. Log. Methods Comput. Sci. **8**(2) (2012). https://doi.org/10.2168/LMCS-8(2:14)2012
60. Urban, C., Tasson, C.: Nominal techniques in Isabelle/HOL. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 38–53. Springer, Heidelberg (2005). https://doi.org/10.1007/11532231_4

Finite Two-Dimensional Proof Systems for Non-finitely Axiomatizable Logics

Vitor Greati[1,2](✉) and João Marcos[1]

[1] Programa de Pós-graduação em Sistemas e Computação & DIMAp,
Universidade Federal do Rio Grande do Norte, Natal, Brazil
vitor.greati.017@ufrn.edu.br, jmarcos@dimap.ufrn.br

[2] Bernoulli Institute, University of Groningen, Groningen, The Netherlands

Abstract. The characterizing properties of a proof-theoretical presentation of a given logic may hang on the choice of proof formalism, on the shape of the logical rules and of the sequents manipulated by a given proof system, on the underlying notion of consequence, and even on the expressiveness of its linguistic resources and on the logical framework into which it is embedded. Standard (one-dimensional) logics determined by (non-deterministic) logical matrices are known to be axiomatizable by analytic and possibly finite proof systems as soon as they turn out to satisfy a certain constraint of sufficient expressiveness. In this paper we introduce a recipe for cooking up a two-dimensional logical matrix (or B-matrix) by the combination of two (possibly partial) non-deterministic logical matrices. We will show that such a combination may result in B-matrices satisfying the property of sufficient expressiveness, even when the input matrices are not sufficiently expressive in isolation, and we will use this result to show that one-dimensional logics that are not finitely axiomatizable may inhabit finitely axiomatizable two-dimensional logics, becoming, thus, finitely axiomatizable by the addition of an extra dimension. We will illustrate the said construction using a well-known logic of formal inconsistency called **mCi**. We will first prove that this logic is not finitely axiomatizable by a one-dimensional (generalized) Hilbert-style system. Then, taking advantage of a known 5-valued non-deterministic logical matrix for this logic, we will combine it with another one, conveniently chosen so as to give rise to a B-matrix that is axiomatized by a two-dimensional Hilbert-style system that is both finite and analytic.

Keywords: Hilbert-style proof systems · finite axiomatizability · consequence relations · non-deterministic semantics · paraconsistency

1 Introduction

A logic is commonly defined nowadays as a relation that connects collections of formulas from a formal language and satisfies some closure properties. The

V. Greati acknowledges support from CAPES—Finance Code 001 and from the FWF project P 33548. J. Marcos acknowledges support from CNPq.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 640–658, 2022.
https://doi.org/10.1007/978-3-031-10769-6_37

established connections are called consecutions and each of them has two parts, an antecedent and a succedent, the latter often being said to 'follow from' (or to be a consequence of) the former. A logic may be manufactured in a number of ways, in particular as being induced by the set of derivations justified by the rules of inference of a given proof system. There are different kinds of proof systems, the differences between them residing mainly in the shapes of their rules of inference and on the way derivations are built. We will be interested here in Hilbert-style proof systems ('H-systems', for short), whose rules of inference have the same shape of the consecutions of the logic they canonically induce and whose associated derivations consist in expanding a given antecedent by applications of rules of inference until the desired succedent is produced. A remarkable property of an H-system is that the logic induced by it is the least logic containing the rules of inference of the system; in the words of [24], the system constitutes a 'logical basis' for the said logic.

Conventional H-systems, which we here dub 'Set-Fmla H-systems', do not allow for more than one formula in the succedents of the consecutions that they manipulate. Since [23], however, we have learned that the simple elimination of this restriction on H-systems —that is, allowing for sets of formulas rather than single formulas in the succedents— brings numerous advantages, among which we mention: *modularity* (correspondence between rules of inference and properties satisfied by a semantical structure), *analyticity* (control over the resources demanded to produce a derivation), and the automatic generation of analytic proof systems for a wide class of logics specified by sufficiently expressive non-deterministics semantics, with an associated straightforward proof-search procedure [13,18]. Such generalized systems, here dubbed 'Set-Set H-systems', induce logics whose consecutions involve succedents consisting in a collection of formulas, intuitively understood as 'alternative conclusions'.

An H-system $\mathscr{H}$ is said to be an *axiomatization* for a given logic $\mathscr{L}$ when the logic induced by $\mathscr{H}$ coincides with $\mathscr{L}$. A desirable property for an axiomatization is *finiteness*, namely the property of consisting on a finite collection of schematic axioms and rules of inference. A logic having a finite axiomatization is said to be 'finitely based'. In the literature, one may find examples of logics having a quite simple, finite semantic presentation, being, in contrast, not finitely based in terms of Set-Fmla H-systems [21]. These very logics, however, when seen as companions of logics with multiple formulas in the succedent, turn out to be finitely based in terms of Set-Set H-systems [18]. In other words, by updating the underlying proof-theoretical and the logical formalisms, we are able to obtain a finite axiomatization for logics which in a more restricted setting could not be said to be finitely based. We may compare the above mentioned movement to the common mathematical practice of adding dimensions in order to provide better insight on some phenomenon. A well-known example of that is given by the Fundamental Theorem of Algebra, which provides an elegant solution to the problem of determining the roots of polynomials over a single variable, demanding only that real coefficients should be replaced by complex coefficients. Another example, from Machine Learning, is the 'kernel trick' employed in support vector machines: by increasing the dimensionality of the input space, the transformed data points become more easily separable by hyperplanes, making it possible to achieve better results in classification tasks.

It is worth noting that there are logics that fail to be finitely based in terms of Set-Set H-systems. An example of a logic designed with the sole purpose of illustrating this possibility was provided in [18]. One of the goals of the present work is to show that an important logic from the literature of logics of formal inconsistency (LFIs) called **mCi** is also an example of this phenomenon. This logic results from adding infinitely-many axiom schemas to the logic **mbC**, a logic that is obtained by extending positive classical logic with two axiom schemas. Incidentally, along the proof of this result, we will show that **mCi** is the limit of a strictly increasing chain of LFIs extending **mbC** (comparable to the case of $\mathcal{C}_{\mathrm{Lim}}$ in da Costa's hierarchy of increasingly weaker paraconsistent calculi [16]). A natural question, then, is whether we can enrich our technology, in the same vein, in order to provide finite axiomatizations for all these logics. We answer that in the affirmative by means of the two-dimensional frameworks developed in [11,17]. Logics, in this case, connect pairs of collections of formulas. A consecution, in this setting, may be read as involving formulas that are accepted and those that are not, as well as formulas that are rejected and those that are not. 'Acceptance' and 'rejection' are seen, thus, as two orthogonal dimensions that may interact, making it possible, thus, to express more complex consecutions than those expressible in one-dimensional logics. Two-dimensional H-systems, which we call 'Set^2-Set^2 H-systems', generalize Set-Set H-systems so as to manipulate pairs of collections of formulas, canonically inducing two-dimensional logics and constituting logical bases for them. Another goal of the present work is, therefore, to show how to obtain a two-dimensional logic inhabited by a (possibly not finitely based) one-dimensional logic of interest. More than that, the logic we obtain will be finitely axiomatizable in terms of a Set^2-Set^2 analytic H-system. The only requirements is that the one-dimensional logic of interest must have an associated semantics in terms of a finite non-deterministic logical matrix and that this matrix can be combined with another one through a novel procedure that we will introduce, resulting in a two-dimensional non-deterministic matrix (a B-matrix [9]) satisfying a certain condition of sufficient expressiveness [17]. An application of this approach will be provided here in order to produce the first finite and analytic axiomatization of **mCi**.

The paper is organized as follows: Sect. 2 introduces basic terminology and definitions regarding algebras and languages. Section 3 presents the notions of one-dimensional logics and Set-Set H-systems. Section 4 proves that **mCi** is not finitely axiomatizable by one-dimensional H-systems. Section 5 introduces two-dimensional logics and H-systems, and describes the approach to extending a logical matrix to a B-matrix with the goal of finding a finite two-dimensional axiomatization for the logic associated with the former. Section 6 presents a two-dimensional finite analytic H-system for **mCi**. In the final remarks, we highlight some byproducts of our present approach and some features of the resulting proof systems, in addition to pointing to some directions for further research.[1]

[1] Detailed proofs of some results may be found in https://arxiv.org/abs/2205.08920.

2 Preliminaries

A *propositional signature* is a family $\Sigma := \{\Sigma_k\}_{k\in\omega}$, where each Σ_k is a collection of k-ary *connectives*. We say that Σ *is finite* when its base set $\bigcup_{k\in\omega}\Sigma_k$ is finite. A *non-deterministic algebra over* Σ, or simply Σ*-nd-algebra*, is a structure $\mathbf{A} := \langle A, \cdot_{\mathbf{A}}\rangle$, such that A is a non-empty collection of values called the *carrier* of $\mathbf{A}$, and, for each $k \in \omega$ and $\copyright \in \Sigma_k$, the multifunction $\copyright_{\mathbf{A}} : A^k \to \wp(A)$ is the *interpretation of* $\copyright$ *in* $\mathbf{A}$. When Σ and A are finite, we say that $\mathbf{A}$ is *finite*. When the range of all interpretations of $\mathbf{A}$ contains only singletons, $\mathbf{A}$ is said to be a *deterministic algebra over* Σ, or simply a Σ*-algebra*, meeting the usual definition from Universal Algebra [12]. When $\varnothing$ is not in the range of each $\copyright_{\mathbf{A}}$, $\mathbf{A}$ is said to be *total*. Given a Σ-algebra $\mathbf{A}$ and a $\copyright \in \Sigma_1$, we let $\copyright^0_{\mathbf{A}}(x) := x$ and $\copyright^{i+1}_{\mathbf{A}}(x) := \copyright_{\mathbf{A}}(\copyright^i_{\mathbf{A}}(x))$. A mapping $v : A \to B$ is a *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$ when, for all $k \in \omega$, $\copyright \in \Sigma_k$ and $x_1, \ldots, x_k \in A$, we have $f[\copyright_{\mathbf{A}}(x_1, \ldots, x_k)] \subseteq \copyright_{\mathbf{B}}(f(x_1), \ldots, f(x_k))$. The set of all homomorphisms from $\mathbf{A}$ to $\mathbf{B}$ is denoted by $\mathsf{Hom}_\Sigma(\mathbf{A}, \mathbf{B})$. When $\mathbf{B} = \mathbf{A}$, we write $\mathsf{End}_\Sigma(\mathbf{A})$, rather than $\mathsf{Hom}_\Sigma(\mathbf{A}, \mathbf{A})$, for the set of *endomorphisms on* $\mathbf{A}$.

Let P be a denumerable collection of *propositional variables* and Σ be a propositional signature. The absolutely free Σ-algebra freely generated by P is denoted by $\mathbf{L}_\Sigma(P)$ and called the Σ*-language generated by* P. The elements of $L_\Sigma(P)$ are called Σ*-formulas*, and those among them that are not propositional variables are called Σ*-compounds*. Given $\Phi \subseteq L_\Sigma(P)$, we denote by Φ^{c} the set $L_\Sigma(P)\backslash\Phi$. The homomorphisms from $\mathbf{L}_\Sigma(P)$ to $\mathbf{A}$ are called *valuations on* $\mathbf{A}$, and we denote by $\mathsf{Val}_\Sigma(\mathbf{A})$ the collection thereof. Additionally, endomorphisms on $\mathbf{L}_\Sigma(P)$ are dubbed Σ*-substitutions*, and we let $\mathsf{Subs}^P_\Sigma := \mathsf{End}_\Sigma(\mathbf{L}_\Sigma(P))$; when there is no risk of confusion, we may omit the superscript from this notation.

Given $\varphi \in L_\Sigma(P)$, let $\mathsf{props}(\varphi)$ be the set of propositional variables occurring in φ. If $\mathsf{props}(\varphi) = \{p_1, \ldots, p_k\}$, we say that φ is k-ary (*unary*, for $k = 1$; *binary*, for $k = 2$) and let $\varphi_{\mathbf{A}} : A^k \to \wp(A)$ be *the* k*-ary multifunction on* $\mathbf{A}$ *induced by* φ, where, for all $x_1, \ldots, x_k \in A$, we have $\varphi_{\mathbf{A}}(x_1, \ldots, x_k) := \{v(\varphi) \mid v \in \mathsf{Val}_\Sigma(\mathbf{A})$ and $v(p_i) = x_i$, for $1 \leq i \leq k\}$. Moreover, given $\psi_1, \ldots, \psi_k \in L_\Sigma(P)$, we write $\varphi(\psi_1, \ldots, \psi_k)$ for the Σ-formula $\varphi_{\mathbf{L}_\Sigma(P)}(\psi_1, \ldots, \psi_k)$, and, where $\Phi \subseteq L_\Sigma(P)$ is a set of k-ary Σ-formulas, we let $\Phi(\psi_1, \ldots, \psi_k) := \{\varphi(\psi_1, \ldots, \psi_k) \mid \varphi \in \Phi\}$. Given $\varphi \in L_\Sigma(P)$, by $\mathsf{subf}(\varphi)$ we refer to the set of *subformulas of* φ. Where θ is a unary Σ-formula, we define the set $\mathsf{subf}^\theta(\varphi)$ as $\{\sigma(\theta) \mid \sigma : P \to \mathsf{subf}(\varphi)\}$. Given a set $\Theta \supseteq \{p\}$ of unary Σ-formulas, we set $\mathsf{subf}^\Theta(\varphi) := \bigcup_{\theta\in\Theta}\mathsf{subf}^\theta(\varphi)$. For example, if $\Theta = \{p, \neg p\}$, we will have $\mathsf{subf}^\Theta(\neg(q \vee r)) = \{q, r, q \vee r, \neg(q \vee r)\} \cup \{\neg q, \neg r, \neg(q \vee r), \neg\neg(q \vee r)\}$. Such generalized notion of subformulas will be used in the next section to provide a more generous proof-theoretical concept of *analyticity*.

3 One-Dimensional Consequence Relations

A Set-Set *statement* (or *sequent*) is a pair $(\Phi, \Psi) \in \wp(L_\Sigma(P)) \times \wp(L_\Sigma(P))$, where Φ is dubbed the *antecedent* and Ψ the *succedent*. A *one-dimensional con-*

sequence relation on $L_\Sigma(P)$ is a collection $\rhd$ of Set-Set statements satisfying, for all $\Phi, \Psi, \Phi', \Psi' \subseteq L_\Sigma(P)$,

(O) if $\Phi \cap \Psi \neq \varnothing$, then $\Phi \rhd \Psi$
(D) if $\Phi \rhd \Psi$, then $\Phi \cup \Phi' \rhd \Psi \cup \Psi'$
(C) if $\Pi \cup \Phi \rhd \Psi \cup \Pi^{\mathsf{c}}$ for all $\Pi \subseteq L_\Sigma(P)$, then $\Phi \rhd \Psi$

Properties **(O)**, **(D)** and **(C)** are called *overlap*, *dilution* and *cut*, respectively. The relation $\rhd$ is called *substitution-invariant* when it satisfies, for every $\sigma \in \mathsf{Subs}_\Sigma$,

(S) if $\Phi \rhd \Psi$, then $\sigma[\Phi] \rhd \sigma[\Psi]$

and it is called *finitary* when it satisfies

(F) if $\Phi \rhd \Psi$, then $\Phi^{\mathrm{f}} \rhd \Psi^{\mathrm{f}}$ for some finite $\Phi^{\mathrm{f}} \subseteq \Phi$ and $\Psi^{\mathrm{f}} \subseteq \Psi$

One-dimensional consequence relations will also be referred to as *one-dimensional logics*. Substitution-invariant finitary one-dimensional logics will be called *standard*. We will denote by $\blacktriangleright$ the complement of $\rhd$, called the *compatibility relation associated with* $\rhd$ [10].

A Set-Fmla *statement* is a sequent having a single formula as consequent. When we restrict standard consequence relations to collections of Set-Fmla statements, we define the so-called (substitution-invariant finitary) *Tarskian consequence relations*. Every one-dimensional consequence relation $\rhd$ determines a Tarskian consequence relation $\vdash_{\rhd} \subseteq \mathscr{P}(L_\Sigma(P)) \times L_\Sigma(P)$, dubbed *the* Set-Fmla *Tarskian companion of* $\rhd$, such that, for all $\Phi \cup \{\psi\} \subseteq L_\Sigma(P)$, $\Phi \vdash_{\rhd} \psi$ if, and only if, $\Phi \rhd \{\psi\}$. It is well-known that the collection of all Tarskian consequence relations over a fixed language constitutes a complete lattice under set-theoretical inclusion [25]. Given a set C of such relations, we will denote by $\bigsqcup C$ its supremum in the latter lattice.

We present in what follows two ways of obtaining one-dimensional consequence relations: one semantical, via non-deterministic logical matrices [6], and the other proof-theoretical, via Set-Set Hilbert-style systems [18,23].

A *non-deterministic* Σ*-matrix*, or simply Σ*-nd-matrix*, is a structure $\mathbb{M} := \langle \mathbf{A}, D \rangle$, where $\mathbf{A}$ is a Σ-nd-algebra, whose carrier is the set of *truth-values*, and $D \subseteq A$ is the set of *designated truth-values*. Such structures are also known in the literature as 'PNmatrices' [7]; they generalize the so-called 'Nmatrices' [5], which are Σ-nd-matrices with the restriction that $\mathbf{A}$ must be total. From now on, whenever $X \subseteq A$, we denote $A \backslash X$ by $\overline{X}$. In case $\mathbf{A}$ is deterministic, we simply say that $\mathbb{M}$ is a Σ*-matrix*. Also, $\mathbb{M}$ is said to be *finite* when $\mathbf{A}$ is finite. Every Σ-nd-matrix $\mathbb{M}$ determines a substitution-invariant one-dimensional consequence relation over Σ, denoted by $\rhd_{\mathbb{M}}$, such that $\Phi \rhd_{\mathbb{M}} \Psi$ if, and only if, for all $v \in \mathsf{Val}_\Sigma(\mathbf{A})$, $v[\Phi] \cap \overline{D} \neq \varnothing$ or $v[\Psi] \cap D \neq \varnothing$. It is worth noting that $\rhd_{\mathbb{M}}$ is finitary whenever the carrier of $\mathbf{A}$ is finite (the proof runs very similar to that of the same result for Nmatrices [5, Theorem 3.15]).

A *strong homomorphism* between Σ-matrices $\mathbb{M}_1 := \langle \mathbf{A}_1, D_1 \rangle$ and $\mathbb{M}_2 := \langle \mathbf{A}_2, D_2 \rangle$ is a homomorphism h between $\mathbf{A}_1$ and $\mathbf{A}_2$ such that $x \in D_1$ if, and only if, $h(x) \in D_2$. When there is a surjective strong homomorphism between $\mathbb{M}_1$ and $\mathbb{M}_2$, we have that $\rhd_{\mathbb{M}_1} = \rhd_{\mathbb{M}_2}$.

Now, to the Hilbert-style systems. A (*schematic*) Set-Set *rule of inference* R_{s} is the collection of all substitution instances of the Set-Set statement s, called the *schema* of R_{s}. Each $r \in R_{\mathsf{s}}$ is called a *rule instance of* R_{s}. A (*schematic*) Set-Set *H-system* R is a collection of Set-Set rules of inference. When we constrain the rule instances of R to having only singletons as succedents, we obtain the conventional notion of Hilbert-style system, called here Set-Fmla *H-system.*

An R-*derivation* in a Set-Set H-system R is a rooted directed tree t such that every node is labelled with sets of formulas or with a discontinuation symbol $*$, and in which every non-leaf node (that is, a node with child nodes) n in t is an *expansion of* n *by a rule instance* r of R. This means that the antecedent of r is contained in the label of n and that n has exactly one child node for each formula ψ in the succedent of r. These child nodes are, in turn, labelled with the same formulas as those of n plus the respective formula ψ. In case r has an empty succedent, then n has a single child node labelled with $*$. Here we will consider only *finitary* Set-Set H-systems, in which each rule instance has finite antecedent and succedent. In such cases, we only need to consider finite derivations. Figure 1 illustrates how derivations using only finitary rules of inference may be graphically represented. We denote by $\ell^{\mathsf{t}}(\mathsf{n})$ the label of the node n in the tree t. It is worth observing that, for Set-Fmla H-systems, derivations are linear trees (as rule instances have a single formula in their succedents), or, in other words, just sequences of formulas built by applications of the rule instances, matching thus the conventional definition of Hilbert-style systems.

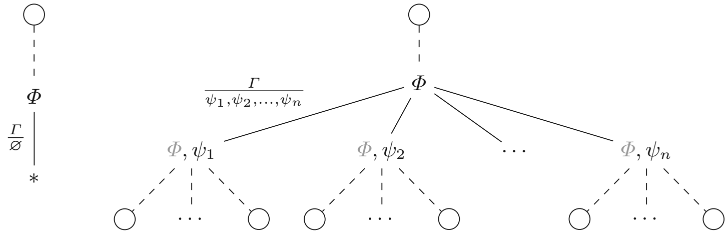

Φ
$\frac{\Gamma}{\varnothing}$
$*$
$\frac{\Gamma}{\psi_1,\psi_2,\ldots,\psi_n}$
Φ
Φ, ψ_1
Φ, ψ_2
$\cdots$
Φ, ψ_n

Fig. 1. Graphical representation of R-derivations, for R finitary. The dashed edges and blank circles represent other branches that may exist in the derivation. We usually omit the formulas inherited from the parent node, exhibiting only the ones introduced by the applied rule of inference. In both cases, we must have $\Gamma \subseteq \Phi$ to enable the application of the rule.

A node n of an R-derivation t is called Δ-*closed* in case it is a leaf node with $\ell^{\mathsf{t}}(\mathsf{n}) = *$ or $\ell^{\mathsf{t}}(\mathsf{n}) \cap \Delta \neq \varnothing$. A branch of t is Δ-closed when it ends in a Δ-closed node. When every branch in t is Δ-closed, we say that R is itself Δ-*closed.* An R-*proof* of a Set-Set statement (Φ, Ψ) is a Ψ-closed R-derivation t such that $\ell^{\mathsf{t}}(\mathsf{rt}(\mathsf{t})) \subseteq \Phi$.

Consider the binary relation $\rhd_{\mathsf{R}}$ on $\mathscr{P}(L_{\Sigma}(P))$ such that $\Phi \rhd_{\mathsf{R}} \Psi$ if, and only if, there is an R-proof of (Φ, Ψ). This relation is the smallest substitution-invariant one-dimensional consequence relation containing the rules of inference of R, and it is finitary when R is finitary. Since Set-Set (and Set-Fmla) H-systems canonically induce one-dimensional consequence relations, we may refer to them as *one-dimensional H-systems* or *one-dimensional axiomatizations*. In case there is a proof of (Φ, Ψ) whose nodes are labelled only with subsets of $\mathsf{subf}^{\Theta}[\Phi \cup \Psi]$, we write $\Phi \rhd_{\mathsf{R}}^{\Theta} \Psi$. In case $\rhd_{\mathsf{R}} = \rhd_{\mathsf{R}}^{\Theta}$, we say that R is *Θ-analytic*. Note that the ordinary notion of analyticity obtains when $\Theta = \{p\}$. From now on, whenever we use the word "analytic" we will mean this extended notion of Θ-analyticity, for some Θ implicit in the context. When the Θ happens to be important for us or we identify any risk of confusion, we will mention it explicitly.

In [13], based on the seminal results on axiomatizability via Set-Set H-systems by Shoesmith and Smiley [23], it was proved that any non-deterministic logical matrix $\mathbb{M}$ satisfying a criterion of sufficient expressiveness is axiomatizable by a Θ-analytic Set-Set Hilbert-style system, which is finite whenever $\mathbb{M}$ is finite, where Θ is the set of separators for the pairs of truth-values of $\mathbb{M}$. According to such criterion, an nd-matrix is *sufficiently expressive* when, for every pair (x, y) of distinct truth-values, there is a unary formula S, called a *separator for* (x, y), such that $\mathrm{S}_{\mathbf{A}}(x) \subseteq D$ and $\mathrm{S}_{\mathbf{A}}(y) \subseteq \overline{D}$, or vice-versa; in other words, when every pair of distinct truth-values is *separable in* $\mathbb{M}$.

We emphasize that it is essential for the above result the adoption of Set-Set H-systems, instead of the more restricted Set-Fmla H-systems. In fact, while two-valued matrices may always be finitely axiomatized by Set-Fmla H-systems [22], there are sufficiently expressive three-valued deterministic matrices [21] and even quite simple two-valued non-deterministic matrices [19] that fail to be finitely axiomatized by Set-Fmla H-systems. When the nd-matrix at hand is not sufficiently expressive, we may observe the same phenomenon of not having a finite axiomatization also in terms of Set-Set H-systems, even if the said nd-matrix is finite. The first example (and, to the best of our knowledge, the only one in the current literature) of this fact appeared in [13], which we reproduce here for later reference:

Example 1. Consider the signature $\Sigma := \{\Sigma_k\}_{k \in \omega}$ such that $\Sigma_1 := \{g, h\}$ and $\Sigma_k := \varnothing$ for all $k \neq 1$. Let $\mathbb{M} := \langle \mathbf{A}, \{\mathbf{a}\} \rangle$ be a Σ-nd-matrix, with $A := \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and

$$g_{\mathbf{A}}(x) = \begin{cases} \{\mathbf{a}\}, & \text{if } x = \mathbf{c} \\ A, & \text{otherwise} \end{cases} \qquad h_{\mathbf{A}}(x) = \begin{cases} \{\mathbf{b}\}, & \text{if } x = \mathbf{b} \\ A, & \text{otherwise} \end{cases}$$

This matrix is not sufficiently expressive because there is no separator for the pair $(\mathbf{b}, \mathbf{c})$, and [13] proved that it is not axiomatizable by a finite Set-Set H-system, even though an infinite Set-Set system that captures it has a quite simple description in terms of the following infinite collection of schemas:

$$\frac{h^i(p)}{p, g(p)}, \text{ for all } i \in \omega.$$

In the next section, we reveal another example of this same phenomenon, this time of the known LFI [14] called **mCi**. In the path of proving that this logic is not axiomatizable by a finite Set-Set H-system, we will show that there are infinitely many LFIs between **mbC** and **mCi**, organized in a strictly increasing chain whose limit is **mCi** itself.

Before continuing, it is worth emphasizing that any given non-sufficiently expressive nd-matrix may be conservatively extended to a sufficiently expressive nd-matrix provided new connectives are added to the language [18]. These new connectives have the sole purpose of separating the pairs of truth-values for which no separator is available in the original language. The Set-Set system produced from this extended nd-matrix can, then, be used to reason over the original logic, since the extension is conservative. However, these new connectives, which a priori have no meaning, are very likely to appear in derivations of consecutions of the original logic. This might not look like an attractive option to inferentialists who believe that purity of the schematic rules governing a given logical constant is essential for the meaning of the latter to be coherently fixed. In the subsequent sections, we will introduce and apply a potentially more expressive notion of logic in order to provide a *finite* and *analytic* H-system for logics that are not finitely axiomatizable in one dimension, while preserving their original languages.

4 The Logic mCi is Not Finitely Axiomatizable

A one-dimensional logic $\rhd$ over Σ is said to be $\neg$-*paraconsistent* when we have $p, \neg p \;\blacktriangleright\; q$, for $p, q \in P$. Moreover, $\rhd$ is $\neg$-*gently explosive* in case there is a collection $\bigcirc(p) \subseteq L_\Sigma(P)$ of unary formulas such that, for some $\varphi \in L_\Sigma(P)$, we have $\bigcirc(\varphi), \varphi \blacktriangleright \varnothing$; $\bigcirc(\varphi), \neg\varphi \blacktriangleright \varnothing$, and, for all $\varphi \in L_\Sigma(P)$, $\bigcirc(\varphi), \varphi, \neg\varphi \rhd \varnothing$. We say that $\rhd$ is a *logic of formal inconsistency (LFI)* in case it is $\neg$-paraconsistent yet $\neg$-gently explosive. In case $\bigcirc(p) = \{\circ p\}$, for $\circ$ a (primitive or composite) *consistency connective*, the logic is said also to be a ***C**-system*. In what follows, let Σ° be the propositional signature such that $\Sigma^\circ_1 := \{\neg, \circ\}$, $\Sigma^\circ_2 := \{\wedge, \vee, \supset\}$, and $\Sigma^\circ_k := \varnothing$ for all $k \notin \{1, 2\}$.

One of the simplest **C**-systems is the logic **mbC**, which was first presented in terms of a Set-Fmla H-system over Σ° obtained by extending any Set-Fmla H-system for positive classical logic ($\mathbf{CPL}^+$) with the following pair of axiom schemas:

(em) $p \vee \neg p$
(bc1) $\circ p \supset (p \supset (\neg p \supset q))$

The logic **mCi**, in turn, is the **C**-system resulting from extending the H-system for **mbC** with the following (infinitely many) axiom schemas [20] (the resulting Set-Fmla H-system is denoted here by $\mathscr{H}_{\mathbf{mCi}}$):

(ci) $\neg\circ p \supset (p \wedge \neg p)$
(ci)$_j$ $\circ\neg^j\circ p$ (for all $0 \le j < \omega$)

A unary connective $\copyright$ is said to constitute a *classical negation* in a one-dimensional logic $\rhd$ extending $\mathbf{CPL}^+$ in case, for all $\varphi, \psi \in L_\Sigma(P)$, $\varnothing \rhd \varphi \vee \copyright(\varphi)$ and $\varnothing \rhd \varphi \supset (\copyright(\varphi) \supset \psi)$. One of the main differences between **mCi** and **mbC** is that an inconsistency connective $\bullet$ may be defined in the former using the paraconsistent negation, instead of a classical negation, by setting $\bullet\varphi := \neg\circ\varphi$ [20].

Both logics above were presented in [15] in ways other than H-systems: via tableau systems, via bivaluation semantics and via possible-translations semantics. In addition, while these logics are known not to be characterizable by a single finite deterministic matrix [20], a characteristic nd-matrix is available for **mbC** [1] and a 5-valued non-deterministic logical matrix is available for **mCi** [2], witnessing the importance of non-deterministic semantics in the study of non-classical logics. Such characterizations, moreover, allow for the extraction of sequent-style systems for these logics by the methodologies developed in [3,4]. Since **mCi**'s 5-valued nd-matrix will be useful for us in future sections, we recall it below for ease of reference.

Definition 1. *Let* $\mathcal{V}_5 := \{f, F, I, T, t\}$ *and* $\mathsf{Y}_5 := \{I, T, t\}$. *Define the* Σ°*-matrix* $\mathbb{M}_{\mathbf{mCi}} := \langle \mathbf{A}_5, \mathsf{Y}_5 \rangle$ *such that* $\mathbf{A}_5 := \langle \mathcal{V}_5, \cdot_{\mathbf{A}_5} \rangle$ *interprets the connectives of* Σ° *according to the following:*

$$\wedge_{\mathbf{A}_5}(x_1, x_2) := \begin{cases} \{f\} & \textit{if either } x_1 \notin \mathsf{Y}_5 \textit{ or } x_2 \notin \mathsf{Y}_5 \\ \{I, t\} & \textit{otherwise} \end{cases}$$

$$\vee_{\mathbf{A}_5}(x_1, x_2) := \begin{cases} \{I, t\} & \textit{if either } x_1 \in \mathsf{Y}_5 \textit{ or } x_2 \in \mathsf{Y}_5 \\ \{f\} & \textit{if } x_1{,}x_2 \notin \mathsf{Y}_5 \end{cases}$$

$$\supset_{\mathbf{A}_5}(x_1, x_2) := \begin{cases} \{I, t\} & \textit{if either } x_1 \notin \mathsf{Y}_5 \textit{ or } x_2 \in \mathsf{Y}_5 \\ \{f\} & \textit{if } x_1 \in \mathsf{Y}_5 \textit{ and } x_2 \notin \mathsf{Y}_5 \end{cases}$$

	f	F	I	T	t
$\neg_{\mathbf{A}_5}$	$\{I,t\}$	$\{T\}$	$\{I,t\}$	$\{F\}$	$\{f\}$

	f	F	I	T	t
$\circ_{\mathbf{A}_5}$	$\{T\}$	$\{T\}$	$\{F\}$	$\{T\}$	$\{T\}$

One might be tempted to apply the axiomatization algorithm of [13] to the finite non-deterministic logical matrix defined above to obtain a finite and analytic Set-Set system for **mCi**. However, it is not obvious, at first, whether this matrix is sufficiently expressive or not (we will, in fact, prove that it is not). In what follows, we will show now **mCi** is actually axiomatizable neither by a finite Set-Fmla H-system (first part), nor by a finite Set-Set H-system (second part); it so happens, thus, that it was not by chance that $\mathcal{H}_{\mathbf{mCi}}$ has been originally presented with infinitely many rule schemas. For the first part, we rely on the following general result:

Theorem 1 ([25], Theorem 2.2.8, adapted). *Let* $\vdash$ *be a standard Tarskian consequence relation. Then* $\vdash$ *is axiomatizable by a finite* Set-Fmla *H-system if, and only if, there is no strictly increasing sequence* $\vdash_0, \vdash_1, \ldots, \vdash_n, \ldots$ *of standard Tarskian consequence relations such that* $\vdash = \bigsqcup_{i \in \omega} \vdash_i$.

In order to apply the above theorem, we first present a family of finite Set-Fmla H-systems that, in the sequel, will be used to provide an increasing sequence of standard Tarskian consequence relations whose supremum is precisely **mCi**. Next, we show that this sequence is stricly increasing, by employing the matrix methodology traditionally used for showing the independence of axioms in a proof system.

Definition 2. *For each $k \in \omega$, let $\mathcal{H}^k_{\mathbf{mCi}}$ be a* Set-Fmla *H-system for positive classical logic together with the schemas (em), (bc1), (ci) and (ci)$_j$, for all $0 \leq j \leq k$.*

Since $\mathcal{H}^k_{\mathbf{mCi}}$ may be obtained from $\mathcal{H}_{\mathbf{mCi}}$ by deleting some (infinitely many) axioms, it is immediate that:

Proposition 1. *For every $k \in \omega$, $\vdash_{\mathcal{H}^k_{\mathbf{mCi}}} \subseteq \vdash_{\mathbf{mCi}}$.*

The way we define the promised increasing sequence of consequence relations in the next result is by taking the systems $\mathcal{H}^k_{\mathbf{mCi}}$ with odd superscripts, namely, we will be working with the sequence $\vdash_{\mathcal{H}^1_{\mathbf{mCi}}}, \vdash_{\mathcal{H}^3_{\mathbf{mCi}}}, \vdash_{\mathcal{H}^5_{\mathbf{mCi}}}, \ldots$ Excluding the cases where k is even will facilitate, in particular, the proof of Lemma 3.

Lemma 1. *For each $1 \leq k < \omega$, let $\vdash_k := \vdash_{\mathcal{H}^{2k-1}_{\mathbf{mCi}}}$. Then $\vdash_1 \subseteq \vdash_2 \subseteq \ldots$, and*

$$\vdash_{\mathbf{mCi}} = \bigsqcup\nolimits_{1 \leq k < \omega} \vdash_k .$$

Finally, we prove that the sequence outlined in the paragraph before Lemma 1 is strictly increasing. In order to achieve this, we define, for each $1 \leq k < \omega$, a Σ°-matrix $\mathbb{M}_k$ and prove that $\mathcal{H}^{2k-1}_{\mathbf{mCi}}$ is sound with respect to such matrix. Then, in the second part of the proof (the "independence part"), we show that, for each $1 \leq k < \omega$, $\mathbb{M}_k$ fails to validate the rule schema (ci)$_j$, for $j = 2k$, which is present in $\mathcal{H}^{2(k+1)-1}_{\mathbf{mCi}}$. In this way, by the contrapositive of the soundness result proved in the first part, we will have (ci)$_j$ provable in $\mathcal{H}^{2(k+1)-1}_{\mathbf{mCi}}$ while unprovable in $\mathcal{H}^{2k-1}_{\mathbf{mCi}}$. In what follows, for any $k \in \omega$, we use k^* to refer to the successor of k.

Definition 3. *Let $1 \leq k < \omega$. Define the $2k^*$-valued Σ°-matrix $\mathbb{M}_k := \langle \mathbf{A}_k, D_k \rangle$ such that $D_k := \{k^* + 1, \ldots, 2k^*\}$ and $\mathbf{A}_k := \langle \{1, \ldots, 2k^*\}, \cdot_{\mathbf{A}_k} \rangle$, the interpretation of Σ° in $\mathbf{A}_k$ given by the following operations:*

$$x \vee_{\mathbf{A}_k} y := \begin{cases} 1 & \textit{if } x, y \in \overline{D_k} \\ k^* + 1 & \textit{otherwise} \end{cases} \qquad x \wedge_{\mathbf{A}_k} y := \begin{cases} k^* + 1 & \textit{if } x, y \in D_k \\ 1 & \textit{otherwise} \end{cases}$$

$$x \supset_{\mathbf{A}_k} y := \begin{cases} 1 & \textit{if } x \in D_k \textit{ and } y \notin \overline{D_k} \\ k^* + 1 & \textit{otherwise} \end{cases}$$

$$\circ_{\mathbf{A}_k} x := \begin{cases} 1 & \textit{if } x = 2k^* \\ k^* + 1 & \textit{otherwise} \end{cases} \quad \neg_{\mathbf{A}_k} x := \begin{cases} k^* + 1 & \textit{if } x \in \{1, 2k^*\} \\ x + k^* & \textit{if } 2 \leq x \leq k^* \\ x - (k^* - 1) & \textit{if } k^* + 1 \leq x \leq 2k^* - 1 \end{cases}$$

Before continuing, we state results concerning this construction, which will be used in the remainder of the current line of argumentation. In what follows, when there is no risk of confusion, we omit the subscript '$\mathbf{A}_k$' from the interpretations to simplify the notation.

Lemma 2. *For all $k \geq 1$ and $1 \leq m \leq 2k$,*

$$\neg^{m}_{\mathbf{A}_k}(k^* + 1) = \begin{cases} (k^* + 1) + \frac{m}{2}, & \textit{if } m \textit{ is even} \\ 1 + \frac{m+1}{2}, & \textit{otherwise} \end{cases}$$

Lemma 3. *For all $1 \leq k < \omega$, we have $\vdash_{\mathcal{H}^{2k^*-1}_{\mathbf{mCi}}} \circ\neg^{2k}\circ p$ but $\nvdash_{\mathcal{H}^{2k-1}_{\mathbf{mCi}}} \circ\neg^{2k}\circ p$.*

Finally, Theorem 1, Lemma 1 and Lemma 3 give us the main result:

Theorem 2. **mCi** *is not axiomatizable by a finite* Set-Fmla *H-system.*

For the second part —namely, that no finite Set-Set H-system axiomatizes **mCi**—, we make use of the following result:

Theorem 3 **([23], Theorem 5.37, adapted).** *Let $\rhd$ be a one-dimensional consequence relation over a propositional signature containing the binary connective $\vee$. Suppose that the* Set-Fmla *Tarskian companion of $\rhd$, denoted by $\vdash_{\rhd}$, satisfies the following property:*

$$\Phi, \varphi \vee \psi \vdash_{\rhd} \gamma \textit{ if, and only if, } \Phi, \varphi \vdash_{\rhd} \gamma \textit{ and } \Phi, \psi \vdash_{\rhd} \gamma \qquad \text{(Disj)}$$

If a Set-Set *H-system* R *axiomatizes $\rhd$, then* R *may be converted into a* Set-Fmla *H-system for $\vdash_{\rhd}$ that is finite whenever* R *is finite.*

It turns out that:

Lemma 4. **mCi** *satisfies (Disj).*

Proof. The non-deterministic semantics of **mCi** gives us that, for all $\varphi, \psi \in L_{\Sigma^{\circ}}(P)$, $\varphi \rhd_{\mathbb{M}_{\mathbf{mCi}}} \varphi \vee \psi$; $\psi \rhd_{\mathbb{M}_{\mathbf{mCi}}} \varphi \vee \psi$, and $\varphi \vee \psi \rhd_{\mathbb{M}_{\mathbf{mCi}}} \varphi, \psi$, and such facts easily imply (Disj).

Theorem 4. **mCi** *is not axiomatizable by a finite* Set-Set *H-system.*

Proof. If R were a finite Set-Set H-system for **mCi**, then, by Lemma 4 and Theorem 3, it could be turned into a finite Set-Fmla H-system for this very logic. This would contradict Theorem 2.

Finding a finite one-dimensional H-system for **mCi** (analytic or not) over the same language, then, proved to be impossible. The previous result also tells us that there is no sufficiently expressive non-deterministic matrix that characterizes **mCi** (for otherwise the recipe in [13] would deliver a finite analytic Set-Set H-system for it), and we may conclude, in particular, that:

Corollary 1. *The nd-matrix* $\mathbb{M}_{\mathbf{mCi}}$ *is not sufficiently expressive.*

The pairs of truth-values of $\mathbb{M}_{\mathbf{mCi}}$ that seem not to be separable (at least one of these pairs must not be, in view of the above corollary) are (t, T) and (f, F). The insufficiency of expressive power to take these specific pairs of values apart, however, would be circumvented if we had considered instead the matrix defined below, obtained from $\mathbb{M}_{\mathbf{mCi}}$ by changing its set of designated values:

Definition 4. *Let* $\mathbb{M}^{\mathsf{n}}_{\mathbf{mCi}} := \langle \mathbf{A}_5, \mathsf{N}_5 \rangle$*, where* $\mathsf{N}_5 := \{f, I, T\}$.

Note that, in $\mathbb{M}^{\mathsf{n}}_{\mathbf{mCi}}$, we have $t \notin \mathsf{N}_5$, while $T \in \mathsf{N}_5$, and we have that $f \in \mathsf{N}_5$, while $F \notin \mathsf{N}_5$. Therefore, the single propositional variable p separates in $\mathbb{M}^{\mathsf{n}}_{\mathbf{mCi}}$ the pairs (t, T) and (f, F). On the other hand, it is not clear now whether the pairs (t, F) and (f, T) are separable in this new matrix. Nonetheless, we will see, in the next section, how we can take advantage of the semantics of non-deterministic B-matrices in order to combine the expressiveness of $\mathbb{M}_{\mathbf{mCi}}$ and $\mathbb{M}^{\mathsf{n}}_{\mathbf{mCi}}$ in a very simple and intuitive manner, preserving the language and the algebra shared by these matrices. The notion of logic induced by the resulting structure will not be one-dimensional, as the one presented before, but rather two-dimensional, in a sense we shall detail in a moment. We identify two important aspects of this combination: first, the logics determined by the original matrices can be fully recovered from the combined logic; and, second, since the notions of H-systems and sufficient expressiveness, as well as the axiomatization algorithm of [13], were generalized in [17], the resulting two-dimensional logic may be algorithmically axiomatized by an *analytic* two-dimensional H-system that is *finite* if the combining matrices are finite, provided the criterion of sufficient expressiveness is satisfied after the combination. This will be the case, in particular, when we combine $\mathbb{M}_{\mathbf{mCi}}$ and $\mathbb{M}^{\mathsf{n}}_{\mathbf{mCi}}$. Consequently, this novel way of combining logics provides a quite general approach for producing finite and analytic axiomatizations for logics determined by non-deterministic logical matrices that fail to be finitely axiomatizable in one dimension; this includes the logics from Example 1, and also **mCi**.

5 Two-Dimensional Logics

From now on, we will employ the symbols Y, $\mathsf{⅄}$, N and $\mathsf{И}$ to informally refer to, respectively, the cognitive attitudes of *acceptance*, *non-acceptance*, *rejection* and *non-rejection*, collected in the set $\mathsf{Atts} := \{\mathsf{Y}, \mathsf{⅄}, \mathsf{N}, \mathsf{И}\}$. Given a set $\Phi \subseteq L_{\Sigma}(P)$, we will write Φ_{α} to intuitively mean that a given agent entertains the cognitive attitude $\alpha \in \mathsf{Atts}$ with respect to the formulas in Φ, that is: the formulas in Φ_{Y} will be understood as being accepted by the agent; the ones in $\Phi_{\mathsf{⅄}}$, as non-accepted; the ones in Φ_{N}, as rejected; and the ones in $\Phi_{\mathsf{И}}$, as non-rejected. Where $\alpha \in \mathsf{Atts}$, we let $\tilde{\alpha}$ be its flipped version, that is, $\tilde{\mathsf{Y}} := \mathsf{⅄}$, $\tilde{\mathsf{⅄}} := \mathsf{Y}$, $\tilde{\mathsf{N}} := \mathsf{И}$ and $\tilde{\mathsf{И}} := \mathsf{N}$.

We refer to each $\left(\begin{array}{c|c} \Phi_{\mathsf{И}} & \Phi_{\mathsf{⅄}} \\ \hline \Phi_{\mathsf{Y}} & \Phi_{\mathsf{N}} \end{array}\right) \in \wp(L_{\Sigma}(P))^2 \times \wp(L_{\Sigma}(P))^2$ as a B-*statement*, where $(\Phi_{\mathsf{Y}}, \Phi_{\mathsf{N}})$ is the *antecedent* and $(\Phi_{\mathsf{⅄}}, \Phi_{\mathsf{И}})$ is the *succedent*. The sets in the latter pairs are called *components.* A B-*consequence relation* is a collection $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}$ of B-statements satisfying:

(O2) if $\Phi_{\mathsf{Y}} \cap \Phi_{\text{⅄}} \neq \varnothing$ or $\Phi_{\mathsf{N}} \cap \Phi_{\text{И}} \neq \varnothing$, then $\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}}|\frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}$
(D2) if $\frac{\Psi_{\text{И}}}{\Psi_{\mathsf{Y}}}|\frac{\Psi_{\text{⅄}}}{\Psi_{\mathsf{N}}}$ and $\Psi_\alpha \subseteq \Phi_\alpha$ for every $\alpha \in \mathsf{Atts}$, then $\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}}|\frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}$
(C2) if $\frac{\Omega_{\text{Ƨ}}^{\mathsf{c}}}{\Omega_{\mathsf{S}}}|\frac{\Omega_{\mathsf{S}}^{\mathsf{c}}}{\Omega_{\text{Ƨ}}}$ for all $\Phi_{\mathsf{Y}} \subseteq \Omega_{\mathsf{S}} \subseteq \Phi_{\text{⅄}}^{\mathsf{c}}$ and $\Phi_{\mathsf{N}} \subseteq \Omega_{\text{Ƨ}} \subseteq \Phi_{\text{И}}^{\mathsf{c}}$, then $\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}}|\frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}$

A B-consequence relation is called *substitution-invariant* if, in addition, $\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}}|\frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}$ holds whenever, for every $\sigma \in \mathsf{Subs}_\Sigma$:

(S2) $\frac{\Psi_{\text{И}}}{\Psi_{\mathsf{Y}}}|\frac{\Psi_{\text{⅄}}}{\Psi_{\mathsf{N}}}$ and $\Phi_\alpha = \sigma(\Psi_\alpha)$ for every $\alpha \in \mathsf{Atts}$

Moreover, a B-consequence relation is called *finitary* when it enjoys the property

(F2) if $\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}}|\frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}$, then $\frac{\Phi_{\text{И}}^{\mathsf{f}}}{\Phi_{\mathsf{Y}}^{\mathsf{f}}}|\frac{\Phi_{\text{⅄}}^{\mathsf{f}}}{\Phi_{\mathsf{N}}^{\mathsf{f}}}$, for some finite $\Phi_\alpha^{\mathsf{f}} \subseteq \Phi_\alpha$, and each $\alpha \in \mathsf{Atts}$

In what follows, B-consequence relations will also be referred to as *two-dimensional logics.* The complement of $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}$, sometimes called the *compatibility relation associated with* $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}$ [10], will be denoted by $\frac{\cdot}{\cdot}\nmid\frac{\cdot}{\cdot}$. Every B-consequence relation $\mathsf{C} := \frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}$ induces one-dimensional consequence relations $\rhd_{\mathsf{t}}^{\mathsf{C}}$ and $\rhd_{\mathsf{f}}^{\mathsf{C}}$, such that $\Phi_{\mathsf{Y}} \rhd_{\mathsf{t}}^{\mathsf{C}} \Phi_{\text{⅄}}$ iff $\frac{\varnothing}{\Phi_{\mathsf{Y}}}|\frac{\Phi_{\text{⅄}}}{\varnothing}$, and $\Phi_{\mathsf{N}} \rhd_{\mathsf{f}}^{\mathsf{C}} \Phi_{\text{И}}$ iff $\frac{\Phi_{\text{И}}}{\varnothing}|\frac{\varnothing}{\Phi_{\mathsf{N}}}$. Given a one-dimensional consequence relation $\rhd$, we say that it *inhabits the* t-*aspect of* C if $\rhd = \rhd_{\mathsf{t}}^{\mathsf{C}}$, and that it *inhabits the* f-*aspect of* C if $\rhd = \rhd_{\mathsf{f}}^{\mathsf{C}}$. B-consequence relations actually induce many other (even non-Tarskian) one-dimensional notions of logics; the reader is referred to [9,11] for a thorough presentation on this topic.

As we did for one-dimensional consequence relations, we present now realizations of B-consequence relations, first via the semantics of nd-B-matrices, then by means of two-dimensional H-systems.

A *non-deterministic* B-*matrix over* Σ, or simply Σ-*nd*-B-*matrix*, is a structure $\mathfrak{M} := \langle \mathbf{A}, \mathsf{Y}, \mathsf{N}\rangle$, where $\mathbf{A}$ is a Σ-nd-algebra, $\mathsf{Y} \subseteq A$ is the set of *designated values* and $\mathsf{N} \subseteq A$ is the set of *antidesignated values* of $\mathfrak{M}$. For convenience, we define $\text{⅄} := A \backslash \mathsf{Y}$ to be the set of *non-designated values*, and $\text{И} := A \backslash \mathsf{N}$ to be the set of *non-antidesignated values* of $\mathfrak{M}$. The elements of $\mathsf{Val}_\Sigma(\mathbf{A})$ are dubbed $\mathfrak{M}$-*valuations.* The B-*entailment relation determined by* $\mathfrak{M}$ is a collection $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}\,\mathfrak{M}$ of B-statements such that

(B-ent) $\dfrac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}}\Big|\dfrac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}\,\mathfrak{M}$ iff there is no $\mathfrak{M}$-valuation v such that
$v(\Phi_\alpha) \subseteq \alpha$ for each $\alpha \in \mathsf{Atts}$,

for every $\Phi_{\mathsf{Y}}, \Phi_{\mathsf{N}}, \Phi_{\text{⅄}}, \Phi_{\text{И}} \subseteq L_\Sigma(P)$. Whenever $\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}}|\frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}\,\mathfrak{M}$, we say that the B-statement $\left(\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}}\vdots\frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}\right)$ *holds in* $\mathfrak{M}$ or *is valid in* $\mathfrak{M}$. An $\mathfrak{M}$-valuation that bears witness to $\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}}\nmid\frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}\,\mathfrak{M}$ is called a *countermodel for* $\left(\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}}\vdots\frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}\right)$ *in* $\mathfrak{M}$. One may easily check that $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}\,\mathfrak{M}$ is a substitution-invariant B-consequence relation, that is finitary when A is finite. Taking C as $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}\,\mathfrak{M}$, we define $\rhd_{\mathsf{t}}^{\mathfrak{M}} := \rhd_{\mathsf{t}}^{\mathsf{C}}$ and $\rhd_{\mathsf{f}}^{\mathfrak{M}} := \rhd_{\mathsf{f}}^{\mathsf{C}}$.

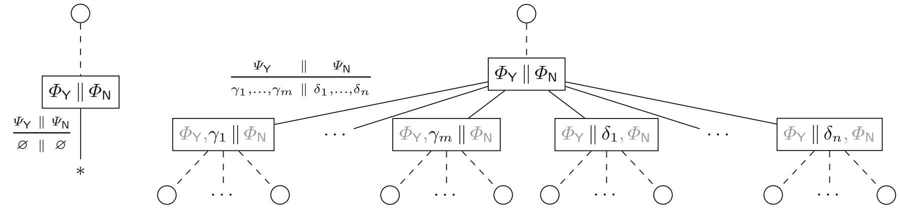

$\Phi_{\mathsf{Y}} \parallel \Phi_{\mathsf{N}}$
$\frac{\Psi_{\mathsf{Y}} \parallel \Psi_{\mathsf{N}}}{\varnothing \parallel \varnothing}$
$*$
$\frac{\Psi_{\mathsf{Y}} \parallel \Psi_{\mathsf{N}}}{\gamma_1,\ldots,\gamma_m \parallel \delta_1,\ldots,\delta_n}$
$\Phi_{\mathsf{Y}} \parallel \Phi_{\mathsf{N}}$
$\Phi_{\mathsf{Y}},\gamma_1 \parallel \Phi_{\mathsf{N}}$
$\cdots$
$\Phi_{\mathsf{Y}},\gamma_m \parallel \Phi_{\mathsf{N}}$
$\Phi_{\mathsf{Y}} \parallel \delta_1, \Phi_{\mathsf{N}}$
$\cdots$
$\Phi_{\mathsf{Y}} \parallel \delta_n, \Phi_{\mathsf{N}}$

Fig. 2. Graphical representation of finite $\mathfrak{R}$-derivations. We emphasize that, in both cases, we must have $\Psi_{\mathsf{Y}} \subseteq \Phi_{\mathsf{Y}}$ and $\Psi_{\mathsf{N}} \subseteq \Phi_{\mathsf{N}}$ to enable the application of the rule.

We move now to two-dimensional, or Set^2-Set^2, H-systems, first introduced in [17]. A *(schematic)* Set^2-Set^2 *rule of inference* $R_{\mathfrak{s}}$ is the collection of all substitution instances of the Set^2-Set^2 statement $\mathfrak{s}$, called the *schema* of $R_{\mathfrak{s}}$. Each $r \in R_{\mathfrak{s}}$ is said to be a *rule instance of* $R_{\mathfrak{s}}$. In a proof-theoretic context, rather than writing the B-statement $\left(\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}} \middle| \frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}\right)$, we shall denote the corresponding rule by $\frac{\Phi_{\mathsf{Y}} \parallel \Phi_{\mathsf{N}}}{\Phi_{\text{⅄}} \parallel \Phi_{\text{И}}}$. A *(schematic)* Set^2-Set^2 *H-system* $\mathfrak{R}$ is a collection of Set^2-Set^2 rules of inference. Set^2-Set^2 *derivations* are as in the Set-Set H-systems, but now the nodes are labelled with pairs of sets of formulas, instead of a single set. When applying a rule instance, each formula in the succedent produces a new branch as before, but now the formula goes to the same component in which it was found in the rule instance. See Fig. 2 for a general representation and compare it with Fig. 1.

Let $\mathfrak{t}$ be an $\mathfrak{R}$-derivation. A node $\mathfrak{n}$ of $\mathfrak{t}$ is $(\Psi_{\text{⅄}}, \Psi_{\text{И}})$*-closed* in case it is discontinued (namely, labelled with $*$) or it is a leaf node with $\ell^{\mathfrak{t}}(\mathfrak{n}) = (\Phi_{\mathsf{Y}}, \Phi_{\mathsf{N}})$ and either $\Phi_{\mathsf{Y}} \cap \Psi_{\text{⅄}} \neq \varnothing$ or $\Phi_{\mathsf{N}} \cap \Psi_{\text{И}} \neq \varnothing$. A branch of $\mathfrak{t}$ is $(\Psi_{\text{⅄}}, \Psi_{\text{И}})$*-closed* when it ends in a $(\Psi_{\text{⅄}}, \Psi_{\text{И}})$-closed node. An $\mathfrak{R}$-derivation $\mathfrak{t}$ is said to be $(\Psi_{\text{⅄}}, \Psi_{\text{И}})$*-closed* when all of its branches are $(\Psi_{\text{⅄}}, \Psi_{\text{И}})$-closed. An $\mathfrak{R}$*-proof* of $\left(\frac{\Phi_{\text{И}}}{\Phi_{\mathsf{Y}}} \middle| \frac{\Phi_{\text{⅄}}}{\Phi_{\mathsf{N}}}\right)$ is a $(\Phi_{\text{⅄}}, \Phi_{\text{И}})$-closed $\mathfrak{R}$-derivation $\mathfrak{t}$ with $\ell^{\mathfrak{t}}(\mathsf{rt}(\mathfrak{t})) \subseteq (\Phi_{\mathsf{Y}}, \Phi_{\mathsf{N}})$. The definitions of the (finitary) substitution-invariant B-consequence relation $\div|\div_{\mathfrak{R}}$ induced by a (finitary) Set^2-Set^2 H-system $\mathfrak{R}$ and Θ-analyticity are obvious generalizations of the corresponding Set-Set definitions.

In [17], the notion of sufficient expressiveness was generalized to nd-B-matrices. We reproduce here the main definitions for self-containment:

Definition 5. *Let* $\mathfrak{M} := \langle \mathbf{A}, \mathsf{Y}, \mathsf{N} \rangle$ *be a* Σ*-nd-*B*-matrix.*

- *Given* $X, Y \subseteq A$ *and* $\alpha \in \{\mathsf{Y}, \mathsf{N}\}$*, we say that* X *and* Y *are* α-separated*, denoted by* $X \#_\alpha Y$*, if* $X \subseteq \alpha$ *and* $Y \subseteq \tilde{\alpha}$*, or vice-versa.*
- *Given distinct truth-values* $x, y \in A$*, a unary formula* S *is a* separator *for* (x, y) *whenever* $\mathrm{S}_{\mathbf{A}}(x) \#_\alpha \mathrm{S}_{\mathbf{A}}(y)$ *for some* $\alpha \in \{\mathsf{Y}, \mathsf{N}\}$*. If there is a separator for each pair of distinct truth-values in* A*, then* $\mathfrak{M}$ *is said to be* sufficiently expressive*.*

In the same work [17], the axiomatization algorithm of [13] was also generalized, guaranteeing that every sufficiently expressive nd-B-matrix $\mathfrak{M}$ is axiomatizable by a Θ-analytic $\textsc{Set}^2$-$\textsc{Set}^2$ H-system, which is finite whenever $\mathfrak{M}$ is finite, where Θ is a set of separators for the pairs of truth-values of $\mathfrak{M}$. Note that, in the second bullet of the above definition, a unary formula is characterized as a separator whenever it separates a pair of truth-values according to *at least one* of the distinguished sets of values. This means that having two of such sets may allow us to separate more pairs of truth-values than having a single set, that is, the nd-B-matrices are, in this sense, potentially more expressive than the (one-dimensional) logical matrices.

Example 2. Let $\mathbf{A}$ be the Σ-nd-algebra from Example 1, and consider the nd-B-matrix $\mathfrak{M} := \langle \mathbf{A}, \{\mathbf{a}\}, \{\mathbf{b}\}\rangle$. As we know, in this matrix the pair $(\mathbf{b}, \mathbf{c})$ is not separable if we consider only the set of designated values $\{\mathbf{a}\}$. However, as we have now the set $\{\mathbf{b}\}$ of antidesignated truth-values, the separation becomes evident: the propositional variable p is a separator for this pair now, since $\mathbf{b} \in \{\mathbf{b}\}$ and $\mathbf{c} \notin \{\mathbf{b}\}$. The recipe from [17] produces the following $\textsc{Set}^2$-$\textsc{Set}^2$ axiomatization for $\mathfrak{M}$, with only three very simple schematic rules of inference:

$$\frac{p \parallel p}{\parallel} \qquad \frac{\parallel}{f(p), p \parallel p} \qquad \frac{\parallel \;\; p}{\parallel t(p)}$$

By construction, the one-dimensional logic determined by the nd-matrix of Example 1 inhabits the t-aspect of $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}_{\mathfrak{M}}$, thus it can be seen as being axiomatized by this *finite* and *analytic* two-dimensional system (contrast with the *infinite* $\textsc{Set}$-$\textsc{Set}$ axiomatization known for this logic provided in that same example).

We constructed above a Σ-nd-B-matrix from two Σ-nd-matrices in such a way that the one-dimensional logics determined by latter are fully recoverable from the former. We formalize this construction below:

Definition 6. *Let* $\mathbb{M} := \langle \mathbf{A}, D\rangle$ *and* $\mathbb{M}' := \langle \mathbf{A}, D'\rangle$ *be* Σ*-nd-matrices. The* B-product *between* $\mathbb{M}$ *and* $\mathbb{M}'$ *is the* Σ*-nd-*B*-matrix* $\mathbb{M} \odot \mathbb{M}' := \langle \mathbf{A}, D, D'\rangle$.

Note that $\Phi \rhd_{\mathbb{M}} \Psi$ iff $\frac{\;}{\Phi}\big|\frac{\Psi}{\;}_{\mathbb{M}\odot\mathbb{M}'}$ iff $\Phi \rhd_{\mathsf{t}}^{\mathbb{M}\odot\mathbb{M}'} \Psi$, and $\Phi \rhd_{\mathbb{M}'} \Psi$ iff $\frac{\Psi}{\;}\big|\frac{\;}{\Phi}_{\mathbb{M}\odot\mathbb{M}'}$ iff $\Phi \rhd_{\mathsf{f}}^{\mathbb{M}\odot\mathbb{M}'} \Psi$. Therefore, $\rhd_{\mathbb{M}}$ and $\rhd_{\mathbb{M}'}$ are easily recoverable from $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}_{\mathbb{M}\odot\mathbb{M}'}$, since they inhabit, respectively, the t-aspect and the f-aspect of the latter. One of the applications of this novel way of putting two distinct logics together was illustrated in that same Example 2 to produce a two-dimensional analytic and finite axiomatization for a one-dimensional logic characterized by a Σ-nd-matrix. As we have shown, the latter one-dimensional logic does not need to be finitely axiomatizable by a $\textsc{Set}$-$\textsc{Set}$ H-system. We present this application of B-products with more generality below:

Proposition 2. *Let* $\mathbb{M} := \langle \mathbf{A}, D\rangle$ *be a* Σ*-nd-matrix and suppose that* $U \subseteq A \times A$ *contains all and only the pairs of distinct truth-values that fail to be separable in* $\mathbb{M}$*. If, for some* $\mathbb{M}' := \langle \mathbf{A}, D'\rangle$*, the pairs in* U *are separable in* $\mathbb{M}'$*, then* $\mathbb{M} \odot \mathbb{M}'$ *is sufficiently expressive (thus, axiomatizable by an analytic* $\textsc{Set}^2$*-*$\textsc{Set}^2$ *H-system, that is finite whenever* $\mathbf{A}$ *is finite).*

6 A Finite and Analytic Proof System for mCi

In the spirit of Proposition 2, we define below a nd-B-matrix by combining the matrices $\mathbb{M}_{\mathbf{mCi}} := \langle \mathbf{A}_5, \mathsf{Y}_5 \rangle$ and $\mathbb{M}^{\mathsf{n}}_{\mathbf{mCi}} := \langle \mathbf{A}_5, \mathsf{N}_5 \rangle$ introduced in Sect. 4 (Definition 1 and Definition 4):

Definition 7. *Let* $\mathfrak{M}_{\mathbf{mCi}} := \mathbb{M}_{\mathbf{mCi}} \odot \mathbb{M}^{\mathsf{n}}_{\mathbf{mCi}} = \langle \mathbf{A}_5, \mathsf{Y}_5, \mathsf{N}_5 \rangle$, *with* $\mathsf{Y}_5 := \{I, T, t\}$ *and* $\mathsf{N}_5 := \{f, I, T\}$.

When we consider now both sets Y_5 and N_5 of designated and antidesignated truth-values, the separation of all truth-values of $\mathbf{A}_5$ becomes possible, that is, $\mathfrak{M}_{\mathbf{mCi}}$ is sufficiently expressive, as guaranteed by Proposition 2. Furthermore, notice that we have two alternatives for separating the pairs (I, t) and (I, T): either using the formula $\neg p$ or the formula $\circ p$. With this finite sufficiently expressive nd-B-matrix in hand, producing a *finite* $\{p, \circ p\}$-analytic two-dimensional H-system for it is immediate by [17, Theorem 2]. Since **mCi** inhabits the t-aspect of $\div\!|\!\div\, \mathfrak{M}_{\mathbf{mCi}}$, we may then conclude that:

Theorem 5. **mCi** *is axiomatizable by a finite and analytic two-dimensional H-system.*

Our axiomatization recipe delivers an H-system with about 300 rule schemas. When we simplify it using the streamlining procedures indicated in that paper, we obtain a much more succinct and insightful presentation, with 28 rule schemas, which we call $\mathfrak{R}_{\mathbf{mCi}}$. The full presentation of this system is given below:

$$\frac{q \;\parallel\;}{p \supset q \;\parallel\;}{\supset}^{\mathbf{mCi}}_1 \quad \frac{\;\parallel\;}{p, p \supset q \;\parallel\;}{\supset}^{\mathbf{mCi}}_2 \quad \frac{p \supset q, p \;\parallel\;}{q \;\parallel\;}{\supset}^{\mathbf{mCi}}_3 \quad \frac{p \;\parallel\;}{q \parallel p \supset q}{\supset}^{\mathbf{mCi}}_4 \quad \frac{p \supset q, \circ(p \supset q) \parallel p \supset q}{\parallel}{\supset}^{\mathbf{mCi}}_5$$

$$\frac{p, q \;\parallel\;}{p \wedge q \;\parallel\;}{\wedge}^{\mathbf{mCi}}_1 \quad \frac{p \wedge q \;\parallel\;}{p \;\parallel\;}{\wedge}^{\mathbf{mCi}}_2 \quad \frac{p \wedge q \;\parallel\;}{q \;\parallel\;}{\wedge}^{\mathbf{mCi}}_3 \quad \frac{\parallel}{p \wedge q \parallel p \wedge q}{\wedge}^{\mathbf{mCi}}_4 \quad \frac{p{\wedge}q, \circ(p{\wedge}q) \parallel p{\wedge}q}{\parallel}{\wedge}^{\mathbf{mCi}}_5$$

$$\frac{p \;\parallel\;}{p \vee q \;\parallel\;}{\vee}^{\mathbf{mCi}}_1 \quad \frac{q \;\parallel\;}{p \vee q \;\parallel\;}{\vee}^{\mathbf{mCi}}_2 \quad \frac{p \vee q \;\parallel\;}{p, q \;\parallel\;}{\vee}^{\mathbf{mCi}}_3 \quad \frac{\parallel}{p, q \parallel p \vee q}{\vee}^{\mathbf{mCi}}_4 \quad \frac{p{\vee}q, \circ(p{\vee}q) \parallel p{\vee}q}{\parallel}{\vee}^{\mathbf{mCi}}_5$$

$$\frac{\circ p \;\parallel\;}{\;\parallel \circ p}{\circ}^{\mathbf{mCi}}_1 \quad \frac{\;\parallel\;}{\circ\circ p \;\parallel\;}{\circ}^{\mathbf{mCi}}_2 \quad \frac{\;\parallel \circ p}{\circ p \;\parallel\;}{\circ}^{\mathbf{mCi}}_3 \quad \frac{\parallel}{\circ p \parallel p}{\circ}^{\mathbf{mCi}}_4 \quad \frac{\parallel}{p \parallel \circ p}{\circ}^{\mathbf{mCi}}_5$$

$$\frac{\parallel}{\parallel \neg p, p}{\neg}^{\mathbf{mCi}}_1 \quad \frac{\neg p, \circ p, p \;\parallel\;}{\parallel}{\neg}^{\mathbf{mCi}}_2 \quad \frac{\neg p, p \;\parallel\;}{\;\parallel p}{\neg}^{\mathbf{mCi}}_3 \quad \frac{\circ\neg p \parallel \neg p, p}{\parallel}{\neg}^{\mathbf{mCi}}_4$$

$$\frac{\;\parallel \neg p, p}{\neg p \;\parallel\;}{\neg}^{\mathbf{mCi}}_5 \quad \frac{\;\parallel\;}{\neg p, \circ p \;\parallel\;}{\neg}^{\mathbf{mCi}}_6 \quad \frac{\;\parallel\;}{\neg p, p \;\parallel\;}{\neg}^{\mathbf{mCi}}_7 \quad \frac{\parallel}{\circ\neg p \parallel p}{\neg}^{\mathbf{mCi}}_8$$

Note that the set of rules $\{ⓒ^{\mathbf{mCi}}_i \mid ⓒ \in \{\wedge, \vee, \supset\}, i \in \{1, 2, 3\}\}$ makes it clear that the t-aspect of the induced B-consequence relation is inhabited by a logic extending positive classical logic, while the remaining rules for these connectives involve interactions between the two dimensions. Also, rule $\neg^{\mathbf{mCi}}_2$ indicates that $\circ$ satisfies one of the main conditions for being taken as a consistency connective in the logic inhabiting the t-aspect. In fact, all these observations are aligned with the fact that the logic inhabiting the t-aspect of $\div\!|\!\div\, \mathfrak{R}_{\mathbf{mCi}}$ is precisely **mCi**. See, in Fig. 3, $\mathfrak{R}_{\mathbf{mCi}}$-derivations showing that, in **mCi**, $\neg\circ p$ and $p \wedge \neg p$ are logically equivalent and that $\circ\neg\circ p$ is a theorem.

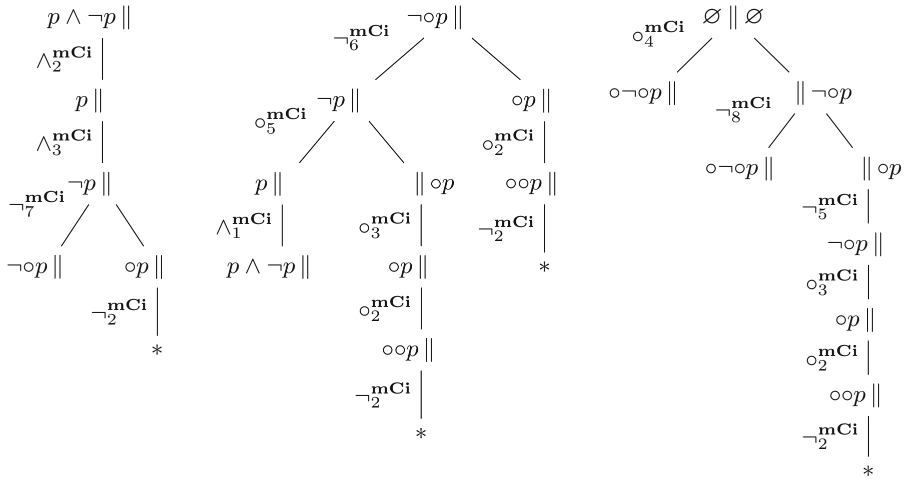


Fig. 3. $\mathfrak{R}_{\mathbf{mCi}}$-derivations showing, respectively, that $\frac{\varnothing}{p\wedge\neg p}\big|\frac{\neg\circ p}{\varnothing}\,\mathfrak{R}_{\mathbf{mCi}}$, $\frac{\varnothing}{\neg\circ p}\big|\frac{p\wedge\neg p}{\varnothing}\,\mathfrak{R}_{\mathbf{mCi}}$ and $\frac{\varnothing}{\varnothing}\big|\frac{\circ\neg\circ p}{\varnothing}\,\mathfrak{R}_{\mathbf{mCi}}$. Note that, for a cleaner presentation, we omit the formulas inherited from parent nodes.

7 Concluding Remarks

In this work, we introduced a mechanism for combining two non-deterministic logical matrices into a non-deterministic B-matrix, creating the possibility of producing finite and analytic two-dimensional axiomatizations for one-dimensional logics that may fail to be finitely axiomatizable in terms of one-dimensional Hilbert-style systems. It is worth mentioning that, as proved in [17], one may perform proof search and countermodel search over the resulting two-dimensional systems in time at most exponential on the size of the B-statement of interest through a straightforward proof-search algorithm.

We illustrated the above-mentioned combination mechanism with two examples, one of them corresponding to a well-known logic of formal inconsistency called **mCi**. We ended up proving not only that this logic is not finitely axiomatizable in one dimension, but also that it is the limit of a strictly increasing chain of LFIs extending the logic **mbC**. From the perspective of the study of B-consequence relations, these examples allow us to eliminate the suspicion that a two-dimensional H-system $\mathfrak{R}$ may always be converted into SET-SET H-systems for the logics inhabiting the one-dimensional aspects of $\frac{\cdot}{\cdot}\big|\frac{\cdot}{\cdot}\,\mathfrak{R}$ without losing any desirable property (in this case, finiteness of the presentation).

At first sight, the formalism of two-dimensional H-systems may be confused with the formalism of n-sided sequents [3,4], in which the objects manipulated by rules of inference (the so-called *n-sequents*) accommodate more than two sets of formulas in their structures. The reader interested in a comparison between these two different approaches is referred to the concluding remarks of [17].

We close with some observations regarding $\mathfrak{M}_{\mathbf{mCi}}$ and the two-dimensional H-system $\mathfrak{R}_{\mathbf{mCi}}$. A one-dimensional logic $\rhd$ is said to be $\neg$-*consistent* when

$\varphi, \neg\varphi \rhd \varnothing$ and $\neg$-*determined* when $\varnothing \rhd \varphi, \neg\varphi$ for all $\varphi \in L_{\Sigma}(P)$. A B-consequence relation $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}$ is said to *allow for gappy reasoning* when $\frac{}{\varphi}\nmid\frac{}{\varphi}$ and to *allow for glutty reasoning* when $\frac{\varphi}{}\nmid\frac{\varphi}{}$, for some $\varphi \in L_{\Sigma}(P)$. Notice that $\neg$-determinedness in the logic inhabiting the t-aspect of a B-consequence relation by no means implies the disallowance of gappy reasoning in the two-dimensional setting: we still have $F \in \overline{\mathsf{Y}_5} \cap \overline{\mathsf{N}_5}$, so one may both non-accept and non-reject a formula φ in $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}\,\mathfrak{R}_{\mathbf{mCi}}$, even though non-accepting both φ and its negation in **mCi** is not possible, in view of rule $\neg_7^{\mathbf{mCi}}$. Similarly, the recovery of $\neg$-consistency achieved via $\circ$ in such logic does not coincide with the gentle disallowance of glutty reasoning in $\frac{\cdot}{\cdot}|\frac{\cdot}{\cdot}\,\mathfrak{R}_{\mathbf{mCi}}$, that is, we do not have, in general, $\frac{}{p, \circ p}\big|\frac{}{p}\,\mathfrak{R}_{\mathbf{mCi}}$ or $\frac{}{p}\big|\frac{}{\circ p, p}\,\mathfrak{R}_{\mathbf{mCi}}$, even though for binary compounds both are derivable in view of rules $©_5^{\mathbf{mCi}}$, for $© \in \{\wedge, \vee, \supset\}$, and $\circ_1^{\mathbf{mCi}}$. With these observations we hope to call attention to the fact that B-consequence relations open the doors for further developments concerning the study of paraconsistency (and, dually, of paracompleteness), as well as the study of recovery operators [8].

References

1. Avron, A.: Non-deterministic matrices and modular semantics of rules. In: Beziau, J.Y. (ed.) Logica Universalis, pp. 149–167. Birkhäuser, Basel (2005). https://doi.org/10.1007/3-7643-7304-0_9
2. Avron, A.: 5-valued non-deterministic semantics for the basic paraconsistent logic mCi. Stud. Log. Grammar Rhetoric 127–136 (2008)
3. Avron, A., Ben-Naim, J., Konikowska, B.: Cut-free ordinary sequent calculi for logics having generalized finite-valued semantics. Log. Univers. **1**, 41–70 (2007). https://doi.org/10.1007/s11787-006-0003-6
4. Avron, A., Konikowska, B.: Multi-valued calculi for logics based on non-determinism. Log. J. IGPL **13**(4), 365–387 (2005). https://doi.org/10.1093/jigpal/jzi030
5. Avron, A., Lev, I.: Non-deterministic multiple-valued structures. J. Log. Comput. **15**(3), 241–261 (2005). https://doi.org/10.1093/logcom/exi001
6. Avron, A., Zamansky, A.: Non-deterministic semantics for logical systems. In: Gabbay, D.M., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. 16, pp. 227–304. Springer, Dordrecht (2011). https://doi.org/10.1007/978-94-007-0479-4_4
7. Baaz, M., Lahav, O., Zamansky, A.: Finite-valued semantics for canonical labelled calculi. J. Autom. Reason. **51**(4), 401–430 (2013). https://doi.org/10.1007/s10817-013-9273-x
8. Barrio, E.A., Carnielli, W.: Volume I: recovery operators in logics of formal inconsistency (special issue). Log. J. IGPL **28**(5), 615–623 (2019). https://doi.org/10.1093/jigpal/jzy053
9. Blasio, C.: Revisitando a lógica de Dunn-Belnap. Manuscrito **40**, 99–126 (2017). https://doi.org/10.1590/0100-6045.2017.v40n2.cb
10. Blasio, C., Caleiro, C., Marcos, J.: What is a logical theory? On theories containing assertions and denials. Synthese **198**(22), 5481–5504 (2019). https://doi.org/10.1007/s11229-019-02183-z
11. Blasio, C., Marcos, J., Wansing, H.: An inferentially many-valued two-dimensional notion of entailment. Bull. Sect. Log. **46**(3/4), 233–262 (2017). https://doi.org/10.18778/0138-0680.46.3.4.05

12. Burris, S., Sankappanavar, H.: A Course in Universal Algebra, vol. 91 (1981)
13. Caleiro, C., Marcelino, S.: Analytic calculi for monadic PNmatrices. In: Iemhoff, R., Moortgat, M., de Queiroz, R. (eds.) WoLLIC 2019. LNCS, vol. 11541, pp. 84–98. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-662-59533-6_6
14. Carnielli, W., Marcos, J.: A taxonomy of C-systems. In: Paraconsistency: The logical way to the inconsistent. Taylor and Francis (2002). https://doi.org/10.1201/9780203910139-3
15. Carnielli, W.A., Coniglio, M.E., Marcos, J.: Logics of formal inconsistency. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. 14, 2nd edn., pp. 1–93. Springer, Dordrecht (2007). https://doi.org/10.1007/978-1-4020-6324-4_1
16. Carnielli, W.A., Marcos, J.: Limits for paraconsistent calculi. Notre Dame J. Formal Log. **40**(3), 375–390 (1999). https://doi.org/10.1305/ndjfl/1022615617
17. Greati, V., Marcelino, S., Marcos, J.: Proof search on bilateralist judgments over non-deterministic semantics. In: Das, A., Negri, S. (eds.) TABLEAUX 2021. LNCS (LNAI), vol. 12842, pp. 129–146. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86059-2_8
18. Marcelino, S., Caleiro, C.: Axiomatizing non-deterministic many-valued generalized consequence relations. Synthese **198**(22), 5373–5390 (2019). https://doi.org/10.1007/s11229-019-02142-8
19. Marcelino, S.: An unexpected Boolean connective. Log. Univer. (2021). https://doi.org/10.1007/s11787-021-00280-7
20. Marcos, J.: Possible-translations semantics for some weak classically-based paraconsistent logics. J. Appl. Non-Classical Log. **18**(1), 7–28 (2008). https://doi.org/10.3166/jancl.18.7-28
21. Palasinska, K.: Deductive systems and finite axiomatization properties. Ph.D. thesis, Iowa State University (1994). https://doi.org/10.31274/rtd-180813-12680
22. Rautenberg, W.: 2-element matrices. Stud. Log. **40**(4), 315–353 (1981)
23. Shoesmith, D.J., Smiley, T.J.: Multiple-Conclusion Logic. Cambridge University Press, Cambridge (1978). https://doi.org/10.1017/CBO9780511565687
24. Wójcicki, R.: Some remarks on the consequence operation in sentential logics. Fundam. Math. **68**, 269–279 (1970)
25. Wójcicki, R.: Theory of Logical Calculi. Synthese Library, 1 edn., Springer, Dordrecht (1988). https://doi.org/10.1007/978-94-015-6942-2

Vampire Getting Noisy: Will Random Bits Help Conquer Chaos? (System Description)

Martin Suda(✉)

Czech Technical University in Prague, Prague, Czech Republic
martin.suda@cvut.cz

Abstract. Treating a saturation-based automatic theorem prover (ATP) as a Las Vegas randomized algorithm is a way to illuminate the chaotic nature of proof search and make it amenable to study by probabilistic tools. On a series of experiments with the ATP Vampire, the paper showcases some implications of this perspective for prover evaluation.

Keywords: Saturation-based proving · Evalutation · Randomization

1 Introduction

Saturation-based proof search is known to be fragile. Even seemingly insignificant changes in the search procedure, such as shuffling the order in which input formulas are presented to the prover, can have a huge impact on the prover's running time and thus on the ability to find a proof within a given time limit.

This *chaotic* aspect of the prover behaviour is relatively poorly understood, yet has obvious consequences for evaluation. A typical experimental evaluation of a new technique T compares the number of problems solved by a baseline run with a run enhanced by T (over an established benchmark and with a fixed timeout). While a higher number of problems solved by the run enhanced by T indicates a benefit of the new technique, it is hard to claim that a certain problem P is getting solved *thanks* to T. It might be that T just helps the prover get lucky on P by a complicated chain of cause and effect not related to the technique T—and the original idea behind it—in any reasonable sense.

We propose to expose and counter the effect of chaotic behaviours by deliberately *injecting randomness* into the prover and observing the results of many independently seeded runs. Although computationally more costly than standard evaluation, such an approach promises to bring new insights. We gain the ability to apply the tools of probability theory and statistics to analyze the results, assign confidences, and single out those problems that *robustly* benefit

This work was supported by the Czech Science Foundation project 20-06390Y and the project RICAIP no. 857306 under the EU-H2020 programme.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 659–667, 2022.
https://doi.org/10.1007/978-3-031-10769-6_38

from the evaluated technique. At the same time, by observing the changes in the corresponding runtime distributions we can even meaningfully establish the effect of the new technique on a single problem in isolation, something that is normally inconclusive due to the threat of chaotic fluctuations.

In this paper, we report on several experiments with a randomized version of the ATP Vampire [9]. After explaining the method in more detail (Sect. 2), we first demonstrate the extent in which the success of a typical Vampire proof search strategy can be ascribed to chance (Sect. 3). Next, we use the collected data to highlight the specifics of comparing two strategies probabilistically (Sect. 4). Finally, we focus on a single problem to see a chaotic behaviour smoothened into a distribution with a high variance (Sect. 5). The paper ends with an overview of related work (Sect. 6) and a discussion (Sect. 7).

2 Randomizing Out Chaos

Any developer of a saturation-based prover will confirm that the behaviour of a specific proving strategy on a specific problem is extremely hard to predict, that a typical experimental evaluation of a new technique (such as the one described earlier) invariably leads to both gains and losses in terms of the solved problems, and that a closer look at any of the "lost" problems often reveals just a complicated chain of cause and effect that steers the prover away from the original path (rather than a simple opportunity to improve the technique further).

These observations bring indirect evidence that the prover's behaviour is chaotic: A specific prover run can be likened to a single bead falling down through the pegs of the famous Galton board[1]. The bead follows a deterministic trajectory, but only because the code fixes every single detail of the execution, including many which the programmer did not care about and which were left as they are merely out of coincidence. We put forward here that any such fixed detail (which does not contribute to an officially implemented heuristic) represents a candidate location for randomization, since a different programmer could have fixed the detail differently and we would still call the code essentially the same.

Implementation: We implemented randomization on top of Vampire version 4.6.1; the code is available as a separate git branch[2]. We divided the randomization opportunities into three groups (governed by three new Vampire options).

Shuffling the input (`-si on`) randomly reorders the input formulas and, recursively, sub-formulas under commutative logical operations. This is done several times throughout the preprocessing pipeline, at the end of which a finished clause normal form is produced. Randomizing traversals (`-rtra on`) happens during saturation and consists of several randomized reorderings including: reordering literals in a newly generated clause and in each given clause before activation, and shuffling the order in which generated clauses are put into the

[1] https://en.wikipedia.org/wiki/Galton_board.

[2] https://github.com/vprover/vampire/tree/randire.

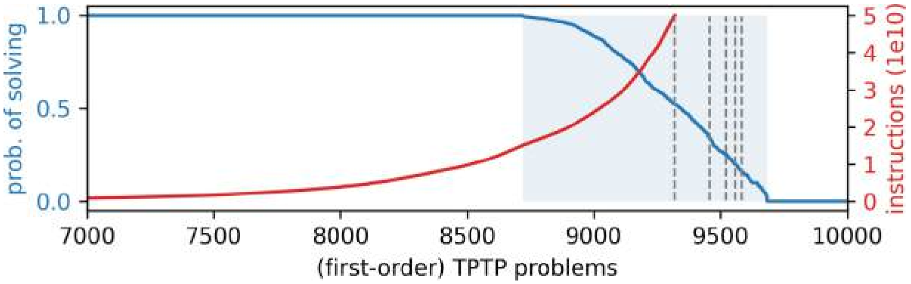


Fig. 1. Blue: first-order TPTP problems ordered by the decreasing probability of being solved by the `dis10` strategy within 50 billion instruction limit. Red: a cactus plot for the same strategy, showing the dependence between a given instruction budget (y-axis) and the number of problems on average solved within that budget (x-axis). (Color figure online)

passive set. It also (partially) randomizes term ids, which are used as tiebreakers in various term indexing operations and determine the default orientation of equational literals in the term sharing structure. Finally, "randomized age-weight ratio" (`-rawr on`) swaps the default, deterministic mechanism for choosing the next queue to select the given clause from [13] for a randomized one (which only respects the age-weight ratio probabilistically).

All the three options were active by default during our experiments.

3 Experiment 1: A Single-Strategy View

First, we set out to establish to what degree the performance of a Vampire strategy can be affected by randomization. We chose the default strategy of the prover except for the saturation algorithm, which we set to Discount, and the age-weight ratio, set to 1:10 (calling the strategy `dis10`). We ran our experiment on the first-order problems from the TPTP library [15] version 7.5.0[3].

To collect our data, we repeatedly (with different seeds) ran the prover on the problems, performing full randomization. We measured the executed instructions[4] needed to successfully solve a problem and used a limit of 50 billion instructions (which roughly corresponds to 15 s of running time on our machine[5]) after which a run was declared unsuccessful. We ran the prover 10 times on each problem and additionally as many times as required to observe the instruction count average (over both successful and unsuccessful runs) stabilize within 1% from any of its 10 previously recorded values[6].

A summary view of the experiment is given by Fig. 1. The most important to notice is the shaded region there, which spans 965 problems that were solved by

[3] Materials accompanying the experiments can be found at https://bit.ly/3JDCwea.

[4] As measured via the `perf_event_open` Linux performance monitoring feature.

[5] A server with Intel(R) Xeon(R) Gold 6140 CPUs @ 2.3 GHz and 500 GB RAM.

[6] Utilizing all the 72 cores of our machine, such data collection took roughly 12 h.

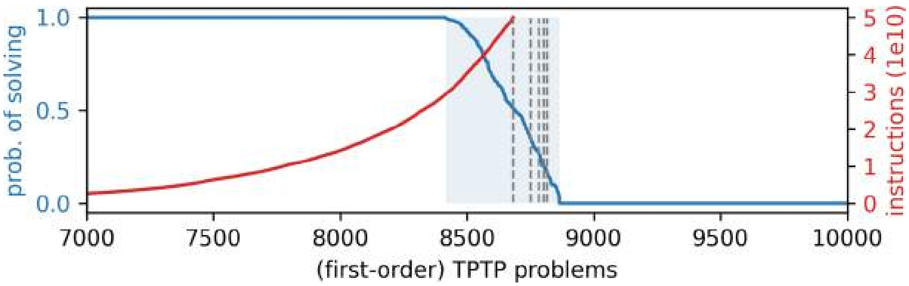


Fig. 2. The effect of turning AVATAR off in the `dis10` strategy (cf. Figure 1).

`dis10` at least once but not by every run. In other words, these problems have probability p of being solved between $0 < p < 1$. This is a relatively large number and can be compared to the 8720 "easy" problems solved by every run. The collected data implies that 9319.1 problems are being solved on average (marked by the left-most dashed line in Fig. 1) with a standard deviation $\sigma = 11.7$. The latter should be an interesting indicator for prover developers: beating a baseline by only 12 TPTP problems can easily be ascribed just to chance.

Figure 1 also contains the obligatory "cactus plot" (explained in the caption), which—thanks to the collected data—can be constructed with the "on average" qualifier. By definition, the plot reaches the left-most dashed line for the full instruction budged of 50 billion. The subsequent dashed lines mark the number of problems we would on average expect to solve by running the prover (independently) on each problem twice, three, four and five times. This is an information relevant for strategy scheduling: e.g., one can expect to solve whole additional 137 problems by running randomized `dis10` for a second time.

Not every strategy exhibits the same degree of variability under randomization. Observe Fig. 2 with a plot analogous to Fig. 1, but for `dis10` in which the AVATAR [16] has been turned off. The shaded area there is now much smaller (and only spans 448 problems). The powerful AVATAR architecture is getting convicted of making proof search more fragile and the prover less robust[7].

Remark. Randomization incurs a small but measurable computational overhead. On a single run of `dis10` over the first-order TPTP (filtering out cases that took less than 1 s to finish, to prevent distortion by rounding errors) the observed median relative time spent randomizing on a single problem was 0.47%, the average 0.59%, and the worse[8] 13.86%. Without randomization, the `dis10` strategy solved 9335 TPTP problems under the 50 billion instruction limit, i.e., 16 problems more than the average reported above. Such is the price we pay for turning our prover into a Las Vegas randomized algorithm.

[7] Another example of a strong but fragile heuristic is the lookahead literal selection [5], which selects literals in a clause based on the current content of the active set: `dis10` enhanced with lookahead solves 9512.4 (±13.8) TPTP problems on average, 8672 problems with $p = 1$ and additional 1382 (!) problems with $0 < p < 1$.

[8] On the hard-to-parse, trivial-to-solve `HWV094-1` with 361 199 clauses.

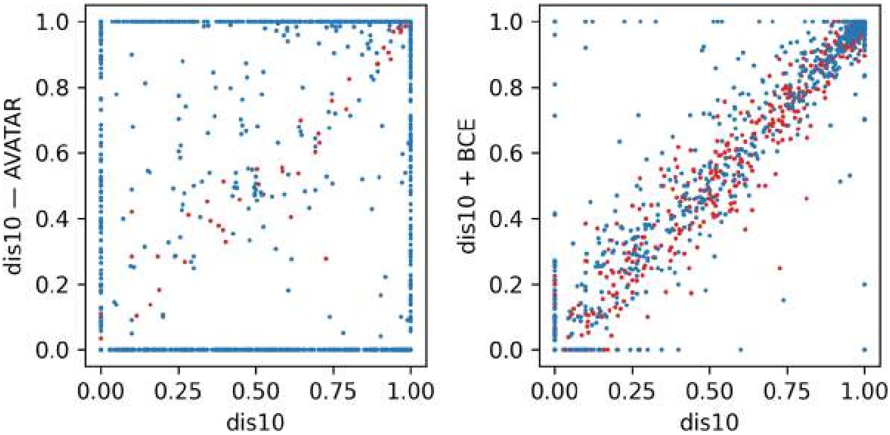


Fig. 3. Scatter plots comparing probabilities of solving a TPTP problem by the baseline `dis10` strategy and 1) `dis10` with AVATAR turned off (left), and 2) `dis10` with blocked clause elimination turned on (right). On problems marked red the respective technique could not be applied (no splittable clauses derived / no blocked clauses eliminated).

4 Experiment 2: Comparing Two Strategies

Once randomized performance profiles of multiple strategies are collected, it is interesting to look at two at a time. Figure 3 shows two very different scatter plots, each comparing our baseline `dis10` to its modified version in terms of the probabilities of solving individual problems.

On the left we see the effect of turning AVATAR off. The technique affects the proving landscape quite a lot and most problems have their mark along the edges of the plot where at least one of the two probabilities has the extreme value of either 0 or 1. What the plot does not show well, is how many marks end up at the extreme corners. These are: 7896 problems easy for both, 661 easy for AVATAR and hard without, 135 hard for AVATAR and easy without.

Such "purified", one-sided gains and losses constitute a new interesting indicator of the impact of a given technique. They should be the first to look at, e.g., during debugging, as they represent the most extreme but robust examples of how the new technique changes the capabilities of the prover.

The right plot is an analogous view, but now at the effect of turning on *blocked clause elimination* (BCE). This is a preprocessing technique coming from the context of propositional satisfiability [7] extended to first-order logic [8]. We see that here most of the visible problems show up as marks along the plot's main diagonal, suggesting a (mostly) negligible effect of the technique. The extreme corners hide: 8648 problems easy for both, 17 easy with BCE (11 satisfiable and 6 unsatisfiable), and 2 easy without BCE (1 satisfiable and 1 unsatisfiable).

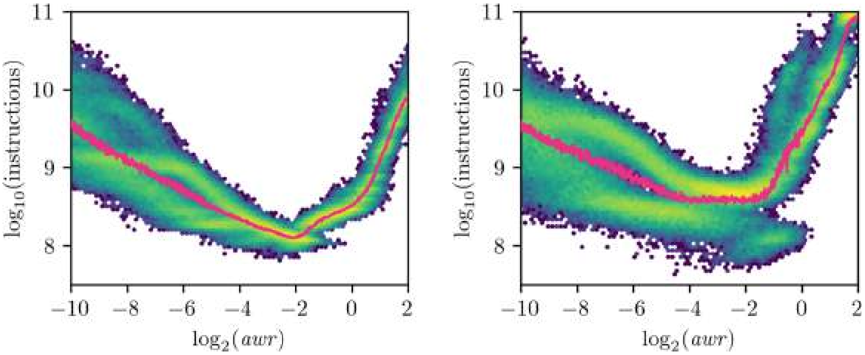


Fig. 4. 2D-histograms for the relative frequencies (color-scale) of how often, given a specific *awr* (x-axis), solving PRO017+2 required the shown number of instructions (y-axis). The curves in pink highlight the mean y-value for every x. The performance of dis10 (left) and the same strategy enhanced by a goal-directed heuristic (right). (Color figure online)

5 Experiment 3: Looking at One Problem at a Time

In their paper on age/weight shapes [13, Fig. 2], Rawson and Reger plot the number of given-clause loops required by Vampire to solve the TPTP problem PRO017+2 as a function of age/weight ratio (*awr*), a ratio specifying how often the prover selects the next clause to activate from its age-ordered and weight-ordered queues, respectively. The curve they obtain is quite "jiggly", indicating a fragile (discontinuous) dependence. Randomization allows us to smoothen the picture and reveal new, until now hidden, (probabilistic) patterns.

The 2D-histogram in Fig. 4 (left) was obtained from 100 independently seeded runs for each of 1200 distinct values of *awr* from between 1:1024 $= 2^{-10}$ and 4:1 $= 2^2$. We can confirm Rawson and Reger's observation of the best *awr* for PRO017+2 lying at around 1:2. However, we can now also attempt to explain the "jiggly-ness" of their curve: With a fragile proof search, even a slight change in *awr* effectively corresponds to an independent sample from the prover's execution resource[9] distribution, which—although changing continuously with *awr*—is of a high variance for our problem (note the log-scale of the y-axis)[10].

The distribution has another interesting property: At least for certain values of *awr* it is distinctly multi-modal. As if the prover can either find a proof quickly (after a lucky event?) or only after much harder effort later and almost nothing in between. Shedding more light on this phenomenon is left for further research.

It is also very interesting to observe the change of such a 2D-histogram when we modify the proof search strategy. Figure 4 (right) shows the effect of turning on SInE-level split queues [3], a goal directed clause selection heuristic

[9] Rawson and Reger [13] counted given-clause loops, we measure instructions.

[10] Even with 100 samples for each value of *awr*, the mean instruction count (rendered in pink in Fig. 4) looks jiggly towards the weight-heavy end of the plot.

(Vampire option `-slsq on`). We can see that the mean instruction count gets worse (for every tried *awr* value) and also the variance of the distribution distinctly increases. A curious effect of this is that we observe the shortest successful runs with `-slsq on`, while we still could not recommend (in the case of `PRO017+2`) this heuristic to the user. The probabilistic view makes us realize that there are competing criteria of prover performance for which one might want to optimize.

6 Related Work

The idea of randomizing a theorem prover is not new. Ertel [2] studied the speedup potential of running independently seeded instances of the connection prover SETHEO [10]. The dashed lines in our Figs. 1 and 2 capture an analogous notion in terms of "additional problems covered" for levels of parallelism 1−5. randoCoP [12] is a randomized version of another connection prover, leanCoP 2.0 [11]: especially in its incomplete setup, several restarts with different seeds helped randoCoP improve over leanCoP in terms of the number of solved problems.

Gomes et al. [4] notice that randomized complete backtracking algorithms for propositional satisfiability (SAT) lead to heavy-tailed runtime distributions on satisfiable instances. While we have not yet analyzed the runtime distributions coming from saturation-based first-order proof search in detail, we definitely observed high variance also for unsatisfiable problems. Also in the domain of SAT, Brglez et al. [1] proposed input shuffling as a way of turning solver's runtime into a random variable and studied the corresponding distributions.

An interesting view on the trade-offs between expected performance of a randomized solver and the risk associated with waiting for an especially long run to finish is given by Huberman et al. [6]. This is related to the last remark of the previous section.

Finally, in the satisfiability modulo theories (SMT) community, input shuffling, or scrambling, has been discussed as an obfuscation measure in competitions [17], where it should prevent the solvers to simply look up a precomputed answer upon recognising a previously seen problem. Notable is also the use of randomization in solver debugging via fuzz testing [14,18].

7 Discussion

As we have seen, the behaviour of a state-of-the-art saturation-based theorem prover is to a considerable degree chaotic and on many problems a mere perturbation of seemingly unimportant execution details decides about the success or the failure of the corresponding run. While this may be seen as a sign of our as-of-yet imperfect grasp of the technology, the author believes that an equally plausible view is that some form of chaos is inherent and originates from the complexity of the theorem proving task itself. (A higher-order logic proof search is expected to exhibit an even higher degree of fragility.)

This paper has proposed randomization as a key ingredient to a prover evaluation method that takes the chaotic nature of proof search into account. The extra cost required by the repeated runs, in itself not unreasonable to pay on contemporary parallel hardware, seems more than compensated by the new insights coming from the probabilistic picture that emerges. Moreover, other uses of randomization are easy to imagine, such as data augmentation for machine learning approaches or the construction of more robust strategy schedules. It feels that we only scratched the surface of the opened-up possibilities. More research will be needed to fully harness the potential of this perspective.

References

1. Brglez, F., Li, X.Y., Stallmann, M.F.M.: On SAT instance classes and a method for reliable performance experiments with SAT solvers. Ann. Math. Artif. Intell. **43**(1), 1–34 (2005). https://doi.org/10.1007/s10472-005-0417-5
2. Ertel, W.: OR-parallel theorem proving with random competition. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624, pp. 226–237. Springer, Heidelberg (1992). https://doi.org/10.1007/BFb0013064
3. Gleiss, B., Suda, M.: Layered clause selection for saturation-based theorem proving. In: Fontaine, P., Korovin, K., Kotsireas, I.S., Rümmer, P., Tourret, S. (eds.) PAAR 7, Paris, France, June-July 2020. CEUR Workshop Proceedings, vol. 2752, pp. 34–52. CEUR-WS.org (2020). http://ceur-ws.org/Vol-2752/paper3.pdf
4. Gomes, C.P., Selman, B., Crato, N., Kautz, H.A.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. Autom. Reason. **24**(1/2), 67–100 (2000). https://doi.org/10.1023/A:1006314320276
5. Hoder, K., Reger, G., Suda, M., Voronkov, A.: Selecting the selection. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 313–329. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_22
6. Huberman, B., Lukose, R., Hogg, T.: An economics approach to hard computational problems. Science **275**, 51–4 (1997)
7. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_10
8. Kiesl, B., Suda, M., Seidl, M., Tompits, H., Biere, A.: Blocked clauses in first-order logic. In: Eiter, T., Sands, D. (eds.) LPAR-21, Maun, Botswana, 7–12 May 2017. EPiC Series in Computing, vol. 46, pp. 31–48. EasyChair (2017)
9. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1–35. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_1
10. Letz, R., Schumann, J., Bayerl, S., Bibel, W.: SETHEO: a high-performance theorem prover. J. Autom. Reason. **8**(2), 183–212 (1992). https://doi.org/10.1007/BF00244282
11. Otten, J.: leanCoP 2.0 and ileanCoP 1.2: high performance lean theorem proving in classical and intuitionistic logic (system descriptions). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 283–291. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71070-7_23
12. Raths, T., Otten, J.: randoCoP: randomizing the proof search order in the connection calculus. In: Konev, B., Schmidt, R.A., Schulz, S. (eds.) PAAR 1, Sydney, Australia, 10–11 August 2008. CEUR Workshop Proceedings, vol. 373. CEUR-WS.org (2008). http://ceur-ws.org/Vol-373/paper-08.pdf

13. Rawson, M., Reger, G.: Old or heavy? decaying gracefully with age/weight shapes. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 462–476. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_27
14. Scott, J., Sudula, T., Rehman, H., Mora, F., Ganesh, V.: BanditFuzz: fuzzing SMT solvers with multi-agent reinforcement learning. In: Huisman, M., Păsăreanu, C., Zhan, N. (eds.) FM 2021. LNCS, vol. 13047, pp. 103–121. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90870-6_6
15. Sutcliffe, G.: The TPTP problem library and associated infrastructure. From CNF to TH0, TPTP v.6.4.0. J. Autom. Reason. **59**(4), 483–502 (2017). https://doi.org/10.1007/s10817-017-9407-7
16. Voronkov, A.: AVATAR: the architecture for first-order theorem provers. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 696–710. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_46
17. Weber, T.: Scrambling and descrambling SMT-LIB benchmarks. In: King, T., Piskac, R. (eds.) SMT@IJCAR 2016, Coimbra, Portugal, 1–2 July 2016. CEUR Workshop Proceedings, vol. 1617, pp. 31–40. CEUR-WS.org (2016). http://ceur-ws.org/Vol-1617/paper3.pdf
18. Winterer, D., Zhang, C., Su, Z.: Validating SMT solvers via semantic fusion. In: Donaldson, A.F., Torlak, E. (eds.) PLDI 2020, London, UK, 15–20 June 2020, pp. 718–730. ACM (2020)

Evolution, Termination, and Decision Problems

On Eventual Non-negativity and Positivity for the Weighted Sum of Powers of Matrices

S. Akshay(✉), Supratik Chakraborty(✉), and Debtanu Pal

Indian Institute of Technology Bombay, Mumbai 400076, India
{akshayss,supratik,debtanu}@cse.iitb.ac.in

Abstract. The long run behaviour of linear dynamical systems is often studied by looking at eventual properties of matrices and recurrences that underlie the system. A basic problem in this setting is as follows: given a set of pairs of rational weights and matrices $\{(w_1, A_1), \ldots, (w_m, A_m)\}$, does there exist an integer N s.t for all $n \geq N$, $\sum_{i=1}^{m} w_i \cdot A_i^n \geq 0$ (resp. > 0). We study this problem, its applications and its connections to linear recurrence sequences. Our first result is that for $m \geq 2$, the problem is as hard as the ultimate positivity of linear recurrences, a long standing open question (known to be coNP-hard). Our second result is that for any $m \geq 1$, the problem reduces to ultimate positivity of linear recurrences. This yields upper bounds for several subclasses of matrices by exploiting known results on linear recurrence sequences. Our third result is a general reduction technique for a large class of problems (including the above) from diagonalizable case to the case where the matrices are simple (have non-repeated eigenvalues). This immediately gives a decision procedure for our problem for diagonalizable matrices.

Keywords: Eventual properties of matrices · Ultimate Positivity · linear recurrence sequences

1 Introduction

The study of eventual or asymptotic properties of discrete-time linear dynamical systems has long been of interest to both theoreticians and practitioners. Questions pertaining to (un)-decidability and/or computational complexity of predicting the long-term behaviour of such systems have been extensively studied over the last few decades. Despite significant advances, however, there remain simple-to-state questions that have eluded answers so far. In this work, we investigate one such problem, explore its significance and links with other known problems, and study its complexity and computability landscape.

This work was partly supported by DST/CEFIPRA/INRIA Project EQuaVE and DST/SERB Matrices Grant MTR/2018/000744.

Author names are in alphabetical order of last names.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 671–690, 2022.
https://doi.org/10.1007/978-3-031-10769-6_39

The time-evolution of linear dynamical systems is often modeled using linear recurrence sequences, or using sequences of powers of matrices. Asymptotic properties of powers of matrices are therefore of central interest in the study of linear differential systems, dynamic control theory, analysis of linear loop programs etc. (see e.g. [26,32,36,37]). The literature contains a rich body of work on the decidability and/or computational complexity of problems related to the long-term behaviour of such systems (see, e.g. [15,19,27,29,36,37]). A question of significant interest in this context is whether the powers of a given matrix of rational numbers eventually have only non-negative (resp. positive) entries. Such matrices, also called *eventually non-negative* (resp. *eventually positive*) matrices, enjoy beautiful algebraic properties ([13,16,25,38]), and have been studied by mathematicians, control theorists and computer scientists, among others. For example, the work of [26] investigates reachability and holdability of non-negative states for linear differential systems – a problem in which eventually non-negative matrices play a central role. Similarly, eventual non-negativity (or positivity) of a matrix modeling a linear dynamical system makes it possible to apply the elegant Perron-Frobenius theory [24,34] to analyze the long-term behaviour of the system beyond an initial number of time steps. Another level of complexity is added if the dynamics is controlled by a set of matrices rather than a single one. For instance, each matrix may model a mode of the linear dynamical system [23]. In a partial observation setting [22,39], we may not know which mode the system has been started in, and hence have to reason about eventual properties of this multi-modal system. This reduces to analyzing the sum of powers of the per-mode matrices, as we will see.

Motivated by the above considerations, we study the problem of determining whether a given matrix of rationals is eventually non-negative or eventually positive and also a generalized version of this problem, wherein we ask if the *weighted sum of powers of a given set of matrices of rationals* is eventually non-negative (resp. positive). Let us formalize the general problem statement. ***Given a set*** $\mathfrak{A} = \{(w_1, A_1), \ldots (w_m, A_m)\}$***, where each*** w_i ***is a rational number and each*** A_i ***is a*** $k \times k$ ***matrix of rationals, we wish to determine if*** $\sum_{i=1}^{m} w_i \cdot A_i^n$ ***has only non-negative (resp. positive) entries for all sufficiently large values of*** n***.*** We call this problem *Eventually Non-Negative (resp. Positive) Weighted Sum of Matrix Powers* problem, or $\mathsf{ENN}_{\mathsf{SoM}}$ (resp. $\mathsf{EP}_{\mathsf{SoM}}$) for short. The eventual non-negativity (resp. positivity) of powers of a single matrix is a special case of the above problem, where $\mathfrak{A} = \{(1, A)\}$. We call this special case the *Eventually Non-Negative (resp. Positive) Matrix* problem, or $\mathsf{ENN}_{\mathsf{Mat}}$ (resp. $\mathsf{EP}_{\mathsf{Mat}}$) for short.

Given the simplicity of the $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$ problem statements, one may be tempted to think that there ought to be simple algebraic characterizations that tell us whether $\sum_{i=1}^{m} w_i \cdot A_i^n$ is eventually non-negative or positive. But in fact, the landscape is significantly nuanced. On one hand, a solution to the general $\mathsf{ENN}_{\mathsf{SoM}}$ or $\mathsf{EP}_{\mathsf{SoM}}$ problem would resolve long-standing open questions in mathematics and computer science. On the other hand, efficient algorithms can indeed be obtained under certain well-motivated conditions. This paper is a study of both these aspects of the problem. Our primary contributions can be summarized as follows. Below, we use $\mathfrak{A} = \{(w_1, A_1), \dots (w_m, A_m)\}$ to define an instance of $\mathsf{ENN}_{\mathsf{SoM}}$ or $\mathsf{EP}_{\mathsf{SoM}}$.

1. If $|\mathfrak{A}| \geq 2$, we show that both $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$ are as hard as the ultimate non-negativity problem for linear recurrence sequences ($\mathsf{UNN}_{\mathsf{LRS}}$, for short). The decidability of $\mathsf{UNN}_{\mathsf{LRS}}$ is closely related to Diophantine approximations, and remains unresolved despite extensive research (see e.g. [31]).
   Since $\mathsf{UNN}_{\mathsf{LRS}}$ is coNP-hard (in fact, as hard as the decision problem for universal theory of reals), so is $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$, when $|\mathfrak{A}| \geq 2$. Thus, unless $\mathsf{P} = \mathsf{NP}$, we cannot hope for polynomial-time algorithms, and any algorithm would also resolve long-standing open problems.
2. On the other hand, regardless of $|\mathfrak{A}|$, we show a reduction in the other direction from $\mathsf{ENN}_{\mathsf{SoM}}$ (resp. $\mathsf{EP}_{\mathsf{SoM}}$) to $\mathsf{UNN}_{\mathsf{LRS}}$ (resp. $\mathsf{UP}_{\mathsf{LRS}}$, the strict version of $\mathsf{UNN}_{\mathsf{LRS}}$). As a consequence, we get decidability and complexity bounds for special cases of $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$, by exploiting recent results on recurrence sequences [30,31,35]. For example, if each matrix A_i in $\mathfrak{A}$ is simple, i.e. has all distinct eigenvalues, we obtain PSPACE algorithms.
3. Finally, we consider the case where A_i is diagonalizable (also called non-defective or inhomogenous dilation map) for each $(w_i, A_i) \in \mathfrak{A}$. This is a practically useful class of matrices and strictly subsumes simple matrices. We present a novel reduction technique for a large family of problems (including eventual non-negativity/positivity, everywhere non-negativity/positivity etc.) over diagonalizable matrices to the corresponding problem over simple matrices. This yields effective decision procedures for $\mathsf{EP}_{\mathsf{SoM}}$ and $\mathsf{ENN}_{\mathsf{SoM}}$ for diagonalizable matrices. Our reduction makes use of a novel perturbation analysis that also has other interesting consequences.

As mentioned earlier, the eventual non-negativity and positivity problem for single rational matrices are well-motivated in the literature, and $\mathsf{EP}_{\mathsf{Mat}}$ (or $\mathsf{EP}_{\mathsf{SoM}}$ with $|\mathfrak{A}| = 1$) is known to be in PTIME [25]. But for $\mathsf{ENN}_{\mathsf{Mat}}$, no decidability results are known to the best of our knowledge. From our work, we obtain two new results about $\mathsf{ENN}_{\mathsf{Mat}}$: (i) in general $\mathsf{ENN}_{\mathsf{Mat}}$ reduces to $\mathsf{UNN}_{\mathsf{LRS}}$ and (ii) for diagonalizable matrices, we can decide $\mathsf{ENN}_{\mathsf{Mat}}$. What is surprising (see Sect. 5) is that the latter decidability result goes via $\mathsf{ENN}_{\mathsf{SoM}}$, i.e. the multiple matrices case. Thus, reasoning about sums of powers of matrices, viz. $\mathsf{ENN}_{\mathsf{SoM}}$, is useful even when reasoning about powers of a single matrix, viz. $\mathsf{ENN}_{\mathsf{Mat}}$.

Potential Applications of $\mathsf{ENN}_{\mathsf{SoM}}$ *and* $\mathsf{EP}_{\mathsf{SoM}}$. A prime motivation for defining the generalized problem statement $\mathsf{ENN}_{\mathsf{SoM}}$ is that it is useful even when reasoning about the single matrix case $\mathsf{ENN}_{\mathsf{Mat}}$. However and unsurprisingly, $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$ are also well-motivated independently. Indeed, for every application involving a linear dynamical system that reduces to $\mathsf{ENN}_{\mathsf{Mat}}/\mathsf{EP}_{\mathsf{Mat}}$, there is a naturally defined aggregated version of the application involving multiple independent linear dynamical systems that reduces to $\mathsf{ENN}_{\mathsf{SoM}}/\mathsf{EP}_{\mathsf{SoM}}$ (e.g., the *swarm of robots* example in [3]).

Beyond this, $\mathsf{ENN}_{\mathsf{SoM}}/\mathsf{EP}_{\mathsf{SoM}}$ arise naturally and directly when solving problems in different practical scenarios. Due to lack of space, we detail two applications here and describe more in the longer version of the paper [3].

Partially Observable Multi-modal Systems. Our first example comes from the domain of cyber-physical systems in a partially observable setting. Consider a system (e.g. a robot) with m modes of operation, where the i^{th} mode dynamics is given by a linear transformation encoded as a $k \times k$ matrix of rationals, say A_i. Thus, if the system state at (discrete) time t is represented by a k-dimensional rational (row) vector $\mathbf{u_t}$, the state at time $t+1$, when operating in mode i, is given by $\mathbf{u_t} A_i$. Suppose the system chooses to operate in one of its various modes at time 0, and then sticks to this mode at all subsequent time. Further, the initial choice of mode is not observable, and we are only given a probability distribution over modes for the initial choice. This is natural, for instance, if our robot (multi-modal system) knows the terrain map and can make an initial choice of which path (mode) to take, but cannot change its path once it has chosen. If p_i is a rational number denoting the probability of choosing mode i initially, then the expected state at time n is given by $\sum_{i=1}^{m} p_i \cdot \mathbf{u_0} A_i^n = \mathbf{u_0}\big(\sum_{i=1}^{m} p_i \cdot A_i^n\big)$. A safety question in this context is whether starting from a state $\mathbf{u_0}$ with all non-negative (resp. positive) components, the system is expected to eventually stay locked in states that have all non-negative (resp. positive) components. In other words, does $\mathbf{u_0}\big(\sum_{i=1}^{m} p_i \cdot A_i^n\big)$ have all non-negative (resp. positive) entries for all sufficiently large n? Clearly, a sufficient condition for an affirmative answer to this question is to have $\sum_{i=1}^{n} p_i \cdot A_i^n$ eventually non-negative (resp. positive), which is an instance of $\mathsf{ENN}_{\mathsf{SoM}}$ (resp. $\mathsf{EP}_{\mathsf{SoM}}$).

Commodity Flow Networks. Consider a flow network where m different commodities $\{c_1, \ldots, c_m\}$ use the same flow infrastructure spanning k nodes, but have different loss/regeneration rates along different links. For every pair of nodes $i, j \in \{1, \ldots, k\}$ and for every commodity $c \in \{c_1, \ldots, c_m\}$, suppose $A_c[i, j]$ gives the fraction of the flow of commodity c starting from i that reaches j through the link connecting i and j (if it exists). In general, $A_c[i, j]$ is the product of the fraction of the flow of commodity c starting at i that is sent along the link to j, and the loss/regeneration rate of c as it flows in the link from i to j. Note that $A_c[i, j]$ can be 0 if commodity c is never sent directly from i to j, or the commodity is lost or destroyed in flowing along the link from i to j. It can be shown that $A_c^n[i, j]$ gives the fraction of the flow of c starting from i that reaches j after n hops through the network. If commodities keep circulating through the network ad-infinitum, we wish to find if the network gets *saturated*, i.e., for all sufficiently long enough hops through the network, there is a non-zero fraction of some commodity that flows from i to j for every pair i, j. This is equivalent to asking if there exists $N \in \mathbb{N}$ such that $\sum_{\ell=1}^{m} A_{c_\ell}^n > 0$. If different commodities have different weights (or costs) associated, with commodity c_i having the weight w_i, the above formulation asks if $\sum_{\ell=1}^{m} w_\ell . A_{c_\ell}^n$ is eventually positive, which is effectively the $\mathsf{EP}_{\mathsf{SoM}}$ problem.

Other Related Work. Our problems of interest are different from other well-studied problems that arise if the system is allowed to choose its mode independently at each time step (e.g. as in Markov decision processes [5,21]). The crucial difference stems from the fact that we require that the mode be chosen once initially, and subsequently, the system must follow the same mode forever. Thus, our problems are prima facie different from those related to general probabilistic or weighted finite automata, where reachability of states and questions pertaining to long-run behaviour are either known to be undecidable or have remained open for long ([6,12,17]). Even in the case of unary probabilistic/weighted finite automata [1,4,8,11], reachability is known in general to be as hard as the Skolem problem on linear recurrences – a long-standing open problem, with decidability only known in very restricted cases. The difference sometimes manifests itself in the simplicity/hardness of solutions. For example, $\mathsf{EP}_{\mathsf{Mat}}$ (or $\mathsf{EP}_{\mathsf{SoM}}$ with $|\mathfrak{A}| = 1$) is known to be in PTIME [25] (not so for $\mathsf{ENN}_{\mathsf{Mat}}$ however), whereas it is still open whether the reachability problem for unary probabilistic/weighted automata is decidable. It is also worth remarking that instead of the sum of powers of matrices, if we considered the product of their powers, we would effectively be solving problems akin to the *mortality problem* [9,10] (which asks whether the all-0 matrix can be reached by multiplying with repetition from a set of matrices) – a notoriously difficult problem. The diagonalizable matrix restriction is a common feature in in the context of linear loop programs (see, e.g., [7,28]), where matrices are used for updates. Finally, logics to reason about temporal properties of linear loops have been studied, although decidability is known only in restricted settings, e.g. when each predicate defines a semi-algebraic set contained in some 3-dimensional subspace, or has intrinsic dimension 1 [20].

2 Preliminaries

The symbols $\mathbb{Q}, \mathbb{R}$, $\mathbb{A}$ and $\mathbb{C}$ denote the set of rational, real, algebraic and complex numbers respectively. Recall that an *algebraic number* is a root of a non-zero polynomial in one variable with rational coefficients. An algebraic number can be real or complex. We use $\mathbb{RA}$ to denote the set of real algebraic numbers (which includes all rationals). The sum, difference and product of two (real) algebraic numbers is again (real) algebraic. Furthermore, every root of a polynomial equation with (real) algebraic coefficients is again (real) algebraic. We call matrices with all rational (resp. real algebraic or real) entries *rational* (resp. *real algebraic* or *real*) *matrices*. We use $A \in \mathbb{Q}^{k\times l}$ (resp. $A \in \mathbb{R}^{k\times l}$ and $A \in \mathbb{RA}^{k\times l}$) to denote that A is a $k \times l$ rational (resp. real and real algebraic) matrix, with rows indexed 1 through k, and columns indexed 1 through l. The entry in the i^{th} row and j^{th} column of a matrix A is denoted $A[i, j]$. If A is a column vector (i.e. $l = 1$), we often use boldface letters, viz. $\mathbf{A}$, to refer to it. In such cases, we use $\mathbf{A}[i]$ to denote the i^{th} component of $\mathbf{A}$, i.e. $A[i, 1]$. The transpose of a $k \times l$ matrix A, denoted A^{T}, is the $l \times k$ matrix obtained by letting $A^{\mathsf{T}}[i, j] = A[j, i]$ for all $i \in \{1, \dots l\}$ and $j \in \{1, \dots k\}$. Matrix A is said to be *non-negative* (resp. *positive*) if all entries of A are non-negative (resp. positive) real numbers. Given a set $\mathfrak{A} = \{(w_1, A_1), \dots (w_m, A_m)\}$ of (weight, matrix) pairs, where each $A_i \in \mathbb{Q}^{k\times k}$ (resp. $\in \mathbb{RA}^{k\times k}$) and each $w_i \in \mathbb{Q}$, we use $\sum \mathfrak{A}^n$ to denote the weighted matrix sum $\sum_{i=1}^{m} w_i \cdot A_i^n$, for every natural number $n > 0$. Note that $\sum \mathfrak{A}^n$ is itself a matrix in $\mathbb{Q}^{k\times k}$ (resp. $\mathbb{RA}^{k\times k}$).

Definition 1. *We say that $\mathfrak{A}$ is eventually non-negative (resp. positive) iff there is a positive integer N s.t., $\sum \mathfrak{A}^n$ is non-negative (resp. positive) for all $n \geq N$.*

The $\mathsf{ENN}_{\mathsf{SoM}}$ (resp. $\mathsf{EP}_{\mathsf{SoM}}$) problem, described in Sect. 1, can now be re-phrased as: *Given a set $\mathfrak{A}$ of pairs of rational weights and rational $k \times k$ matrices, is $\mathfrak{A}$ eventually non-negative (resp. positive)?* As mentioned in Sect. 1, if $\mathfrak{A} = \{(1, A)\}$, the $\mathsf{ENN}_{\mathsf{SoM}}$ (resp. $\mathsf{EP}_{\mathsf{SoM}}$) problem is also called $\mathsf{ENN}_{\mathsf{Mat}}$ (resp. $\mathsf{EP}_{\mathsf{Mat}}$). We note that the study of $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$ with $|\mathfrak{A}| = 1$ is effectively the study of $\mathsf{ENN}_{\mathsf{Mat}}$ and $\mathsf{EP}_{\mathsf{Mat}}$ i.e., wlog we can assume $w = 1$.

The *characteristic polynomial* of a matrix $A \in \mathbb{RA}^{k \times k}$ is given by $det(A - \lambda I)$, where I denotes the $k \times k$ identity matrix. Note that this is a degree k polynomial in λ. The roots of the characteristic polynomial are called the *eigenvalues* of A. The non-zero vector solution of the equation $A\mathbf{x} = \lambda_i \mathbf{x}$, where λ_i is an eigenvalue of A, is called an *eigenvector* of A. Although $A \in \mathbb{RA}^{k \times k}$, in general it can have eigenvalues $\lambda \in \mathbb{C}$ which are all algebraic numbers. An eigenvector is said to be positive (resp. non-negative) if each component of the eigenvector is a positive (resp. non-negative) rational number. A matrix is called *simple* if all its eigenvalues are distinct. Further, a matrix A is called *diagonalizable* if there exists an invertible matrix S and diagonal matrix D such that $SDS^{-1} = A$.

The study of weighted sum of powers of matrices is intimately related to the study of *linear recurrence sequences (LRS)*, as we shall see. We now present some definitions and useful properties of LRS. For more details on LRS, the reader is referred to the work of Everest et al. [14]. A sequence of rational numbers $\langle u \rangle = \langle u_n \rangle_{n=0}^{\infty}$ is called an LRS of *order* k (> 0) if the n^{th} term of the sequence, for all $n \geq k$, can be expressed using the recurrence: $u_n = a_{k-1}u_{n-1} + \ldots + a_1 u_{n-k-1} + a_0 u_{n-k}$. Here, a_0 ($\neq 0$), $a_1, \ldots, a_{k-1} \in \mathbb{Q}$ are called the *coefficients* of the LRS, and $u_0, u_1, \ldots, u_{k-1} \in \mathbb{Q}$ are called the *initial values* of the LRS. Given the coefficients and initial values, an LRS is uniquely defined. However, the same LRS may be defined by multiple sets of coefficients and corresponding initial values. An LRS $\langle u \rangle$ is said to be *periodic* with period ρ if it can be defined by the recurrence $u_n = u_{n-\rho}$ for all $n \geq \rho$. Given an LRS $\langle u \rangle$, its *characteristic polynomial* is $p_{\langle u \rangle}(x) = x^k - \sum_{i=0}^{k-1} a_i x^i$. We can factorize the characteristic polynomial as $p_{\langle u \rangle}(x) = \prod_{j=1}^{d} (x - \lambda_j)^{\rho_j}$, where λ_j is a root, called a *characteristic root* of *algebraic multiplicity* ρ_j. An LRS is called *simple* if $\rho_j = 1$ for all j, i.e. all characteristic roots are distinct. Let $\{\lambda_1, \lambda_2, \ldots, \lambda_d\}$ be distinct roots of $p_{\langle u \rangle}(x)$ with multiplicities $\rho_1, \rho_2, \ldots, \rho_d$ respectively. Then the n^{th} term of the LRS, denoted u_n, can be expressed as $u_n = \sum_{j=1}^{d} q_j(n) \lambda_j^n$, where $q_j(x) \in \mathbb{C}(x)$ are univariate polynomials of degree at most $\rho_j - 1$ with complex coefficients such that $\sum_{j=1}^{d} \rho_j = k$. This representation of an LRS is known as the *exponential polynomial solution* representation. It is well known that scaling an LRS by a constant gives another LRS, and the sum and product of two LRSs is also an LRS (Theorem 4.1 in [14]). Given an LRS $\langle u \rangle$ defined by $u_n = a_{k-1}u_{n-1} + \ldots + a_1 u_{n-k-1} + a_0 u_{n-k}$, we define its *companion matrix* $M_{\langle u \rangle}$ to be the $k \times k$ matrix shown in Fig. 1.

$$M_{\langle u\rangle} = \begin{bmatrix} a_{k-1} & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_2 & 0 & \dots & 1 & 0 \\ a_1 & 0 & \dots & 0 & 1 \\ a_0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Fig. 1. Companion matrix

When $\langle u\rangle$ is clear from the context, we often omit the subscript for clarity of notation, and use M for $M_{\langle u\rangle}$. Let $\mathbf{u} = (u_{k-1}, \dots, u_0)$ be a row vector containing the k initial values of the recurrence, and let $\mathbf{e_k} = (0, 0, \dots 1)^T$ be a column vector of k dimensions with the last element equal to 1 and the rest set to 0s. It is easy to see that for all $n \geq 1$, $\mathbf{u}M^n\mathbf{e_k}$ gives u_n. Note that the eigenvalues of the matrix M are exactly the roots of the characteristic polynomial of the LRS $\langle u\rangle$.

For $\mathbf{u} = (u_{k-1}, \dots, u_0)$, we call the matrix $G_{\langle u\rangle} = \begin{bmatrix} 0 & \mathbf{u} \\ \mathbf{0}^T & M_{\langle u\rangle} \end{bmatrix}$ the *generator matrix* of the LRS $\langle u\rangle$, where $\mathbf{0}$ is a k-dimensional vector of all 0s. We omit the subscript and use G instead of $G_{\langle u\rangle}$, when the LRS $\langle u\rangle$ is clear from the context. It is easy to show from the above that $u_n = G^{n+1}[1, k+1]$ for all $n \geq 0$.

We say that an LRS $\langle u\rangle$ is *ultimately non-negative* (resp. *ultimately positive*) iff there exists $N > 0$, such that $\forall n \geq N$, $u_n \geq 0$ (resp. $u_n > 0$)[1]. The problem of determining whether a given LRS is ultimately non-negative (resp. ultimately positive) is called the *Ultimate Non-negativity* (resp. *Ultimate Positivity*) problem for LRS. We use $\mathsf{UNN_{LRS}}$ (resp. $\mathsf{UP_{LRS}}$) to refer to this problem. It is known [19] that $\mathsf{UNN_{LRS}}$ and $\mathsf{UP_{LRS}}$ are polynomially inter-reducible, and these problems have been widely studied in the literature (e.g., [27,31,32]). A closely related problem is the *Skolem problem*, wherein we are given an LRS $\langle u\rangle$ and we are required to determine if there exists $n \geq 0$ such that $u_n = 0$. The relation between the Skolem problem and $\mathsf{UNN_{LRS}}$ (resp. $\mathsf{UP_{LRS}}$) has been extensively studied in the literature (e.g., [18,19,33]).

3 Hardness of Eventual Non-negativity and Positivity

In this section, we show that $\mathsf{UNN_{LRS}}$ (resp. $\mathsf{UP_{LRS}}$) polynomially reduces to $\mathsf{ENN_{SoM}}$ (resp. $\mathsf{EP_{SoM}}$) when $|\mathfrak{A}| \geq 2$. Since $\mathsf{UNN_{LRS}}$ and $\mathsf{UP_{LRS}}$ are known to be coNP-hard (in fact, as hard as the decision problem for the universal theory of reals Theorem 5.3 [31]), we conclude that $\mathsf{ENN_{SoM}}$ and $\mathsf{EP_{SoM}}$ are also coNP-hard and at least as hard as the decision problem for the universal theory of reals, when $|\mathfrak{A}| \geq 2$. Thus, unless $\mathsf{P} = \mathsf{NP}$, there is no hope of finding polynomial-time solutions to these problems.

Theorem 1. *$\mathsf{UNN_{LRS}}$ reduces to $\mathsf{ENN_{SoM}}$ with $|\mathfrak{A}| \geq 2$ in polynomial time.*

Proof. Given an LRS $\langle u\rangle$ of order k defined by the recurrence $u_n = a_{k-1}u_{n-1} + \dots + a_1u_{n-k-1} + a_0u_{n-k}$ and initial values $u_0, u_1, \dots, u_{k-1}$, construct two

[1] *Ultimately non-negative* (resp. *ultimately positive*) LRS, as defined by us, have also been called *ultimately positive* (resp. *strictly positive*) LRS elsewhere in the literature [31]. However, we choose to use terminology that is consistent across matrices and LRS, to avoid notational confusion.

matrices A_1 and A_2 such that $\langle u\rangle$ is ultimately non-negative iff $(A_1^n + A_2^n)$ is eventually non-negative. Consider $A_1 = \begin{bmatrix} 0 & \mathbf{u} \\ \mathbf{0}^T & M \end{bmatrix}$, the generator matrix of $\langle u\rangle$ and $A_2 = \begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{0}^T & P \end{bmatrix}$, where $P \in \mathbb{Q}^{k\times k}$ is constructed such that : $P[i,j] \geq |M[i,j]|$. For example P can be constructed as: $P[i,j] = M[i,j]$ for all $j \in [2,k]$ and $i \in [1,k]$ and $P[i,j] = max(|a_0|, |a_1|, \ldots, |a_{k-1}|) + 1$ for $j = 1$. Now consider the sequence of matrices defined by $A_1^n + A_2^n$, for all $n \geq 1$. By properties of the generator matrix, it is easily verified that $A_1^n = \begin{bmatrix} 0 & \mathbf{u}M^{n-1} \\ \mathbf{0}^T & M^n \end{bmatrix}$. Similarly, we get $A_2^n = \begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{0}^T & P^n \end{bmatrix}$. Therefore, $A_1^n + A_2^n = \begin{bmatrix} 0 & \mathbf{u}M^{n-1} \\ \mathbf{0}^T & P^n + M^n \end{bmatrix}$, for all $n \geq 1$. Now, we can observe that $P^n + M^n$ is always non-negative, since $P[i,j] \geq |M[i,j]| \geq 0$ for all $i, j \in \{1, \ldots k\}$ and hence $P^n[i,j] + M^n[i,j] \geq 0$ for all $i, j \in \{1, \ldots k\}$ and $n \geq 1$. Thus we conclude that $A(n) = A_1^n + A_2^n \geq 0$ $(n \geq 1)$ iff $\langle u\rangle$ is ultimately non-negative, since the elements $A(n)[1,1] \ldots, A(n)[1,k+1]$ consists of $(u_{n+k-2} \ldots, u_n, u_{n-1})$ and the rest of the elements are non-negative. □

Observe that the same reduction technique works if we are required to use more than 2 matrices in $\mathsf{ENN}_{\mathsf{SoM}}$. Indeed, we can construct matrices $A_3, A_4, \ldots, A_m$ similar to the construction of A_2 in the reduction above, by having the $k \times k$ matrix in the bottom right (see definition of A_2) to have positive values greater than the maximum absolute value of every element in the companion matrix.

A simple modification of the above proof setting $A_2 = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{1}^T & P \end{bmatrix}$, where $\mathbf{1}$ denotes the k-dimensional vector of all 1's gives us the corresponding hardness result for $\mathsf{EP}_{\mathsf{SoM}}$ (see [3] for details).

Theorem 2. *$\mathsf{UP}_{\mathsf{LRS}}$ reduces to $\mathsf{EP}_{\mathsf{SoM}}$ with $|\mathfrak{A}| \geq 2$ in polynomial time.*

We remark that for the reduction technique used in Theorems 1 and 2 to work, we need at least two (weight, matrix) pairs in $\mathfrak{A}$. For explanation of why this reduction doesn't work when $|\mathfrak{A}| = 1$, we refer the reader to [3]. Having shown the hardness of $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$ when $|\mathfrak{A}| \geq 2$, we now proceed to establish upper bounds on the computational complexity of these problems.

4 Upper Bounds on Eventual Non-negativity and Positivity

In this section, we show that $\mathsf{ENN}_{\mathsf{SoM}}$ (resp. $\mathsf{EP}_{\mathsf{SoM}}$) is polynomially reducible to $\mathsf{UNN}_{\mathsf{LRS}}$ (resp. $\mathsf{UP}_{\mathsf{LRS}}$), regardless of $|\mathfrak{A}|$.

Theorem 3. *$\mathsf{ENN}_{\mathsf{SoM}}$, reduces to $\mathsf{UNN}_{\mathsf{LRS}}$ in polynomial time.*

The proof is in two parts. First, we show that for a single matrix A, we can construct a linear recurrence $\langle a\rangle$ such that A is eventually non-negative iff

$\langle a \rangle$ is ultimately non-negative. Then, we show that starting from such a linear recurrence for each matrix in $\mathfrak{A}$, we can construct a new LRS, say $\langle a^{\star} \rangle$, with the property that the weighted sum of powers of the matrices in $\mathfrak{A}$ is eventually non-negative iff $\langle a^{\star} \rangle$ is ultimately non-negative. Our proof makes crucial use of the following property of matrices.

Lemma 1 Adapted from Lemma 1.1 of [19]**).** *Let $A \in \mathbb{Q}^{k \times k}$ be a rational matrix with characteristic polynomial $p_A(\lambda) = det(A - \lambda I)$. Suppose we define the sequence $\langle a^{ij} \rangle$ for every $1 \leq i, j \leq k$ as follows: $a^{i,j}_n = A^{n+1}[i, j]$, for all $n \geq 0$. Then $\langle a^{i,j} \rangle$ is an LRS of order k with characteristic polynomial $p_A(x)$ and initial values given by $a^{ij}_0 = A^1[i, j], \ldots a^{ij}_{k-1} = A^k[i, j]$.*

This follows from the Cayley-Hamilton Theorem and the reader is referred to [19] for further details. From Lemma 1, it is easy to see that the LRS $\langle a^{i,j} \rangle$ for all $1 \leq i, j \leq k$ share the same order and characteristic polynomial (hence the defining recurrence) and differ only in their initial values. For notational convenience, we say that the LRS $\langle a^{i,j} \rangle$ is *generated by* $A[i, j]$.

Proposition 1. *A matrix $A \in \mathbb{Q}^{k \times k}$ is eventually non-negative iff all LRS $\langle a^{i,j} \rangle$ generated by $A[i, j]$ for all $1 \leq i, j \leq k$ are ultimately non-negative.*

The proof follows from the definition of eventually non-negative matrices and the definition of $\langle a^{ij} \rangle$. Next we define the notion of interleaving of LRS.

Definition 2. *Consider a set $S = \{\langle u^i \rangle : 0 \leq i < t\}$ of t LRSes, each having order k and the same characteristic polynomial. An LRS $\langle v \rangle$ is said to be the **LRS-interleaving** of S iff $v_{tn+s} = u^s_n$ for all $n \in \mathbb{N}$ and $0 \leq s < t$.*

Observe that, the order of $\langle v \rangle$ is tk and its initial values are given by the interleaving of the k initial values of the LRSes $\langle u^i \rangle$. Formally, the initial values are $v_{tj+i} = u^i_j$ for $0 \leq i < t$ and $0 \leq j < k$. The characteristic polynomial $p_{\langle v \rangle}(s)$ is equal to $p_{\langle u^i \rangle}(x^t)$.

Proposition 2. *The LRS-interleaving $\langle v \rangle$ of a set of LRSes $S = \{\langle u^i \rangle : 0 \leq i < t\}$ is ultimately non-negative iff each LRS $\langle u^i \rangle$ in S is ultimately non-negative.*

Now, from the definitions of LRSes $\langle a^{i,j} \rangle$, $\langle u^i \rangle$ and $\langle v \rangle$, and from Propositions 1 and 2, we obtain the following crucial lemma.

Lemma 2. *Given a matrix $A \in \mathbb{Q}^{k \times k}$, let $S = \{\langle u^i \rangle \mid u^i_n = a^{pq}_n$, where $p = \lfloor i/k \rfloor + 1,\ q = i \mod k + 1,\ 0 \leq i < k^2\}$ be the set of k^2 LRSes mentioned in Lemma 1. The LRS $\langle v \rangle$ generated by LRS-interleaving of S satisfies the following:*

1. *A is eventually non-negative iff $\langle v \rangle$ is ultimately non-negative.*
2. *$p_{\langle v \rangle}(x) = \prod_{i=1}^{k}(x^{k^2} - \lambda_i)$, where $\lambda_1, \ldots \lambda_k$ are the (possibly repeated) eigenvalues of A.*
3. *$v_{rk^2+sk+t} = u^{sk+t}_r = a^{s+1,t+1}_r = A^{r+1}[s + 1, t + 1]$ for all $r \in \mathbb{N}$, $0 \leq s, t < k$.*

We lift this argument from a single matrix to a weighted sum of matrices.

Lemma 3. *Given $\mathfrak{A} = \{(w_1, A_1), \ldots, (w_m, A_m)\}$, there exists a linear recurrence $\langle a^\star \rangle$, such that $\sum_{i=1}^m w_i A_i^n$ is eventually non-negative iff $\langle a^\star \rangle$ is ultimately non-negative.*

Proof. For each matrix A_i in $\mathfrak{A}$, let $\langle v^i \rangle$ be the interleaved LRS as constructed in Lemma 2. Let $w_i \langle v^i \rangle$ denote the scaled LRS whose n^{th} entry is $w_i v_n^i$ for all $n \geq 0$. The LRS $\langle a^\star \rangle$ is obtained by adding the scaled LRSes $w_1 \langle v^1 \rangle, w_2 \langle v^2 \rangle, \ldots$ $w_m \langle v^m \rangle$. Clearly, $a_n^\star$ is non-negative iff $\sum_{i=1}^m w_i v_n^i$ is non-negative. From the definition of v^i (see Lemma 2), we also know that for all $n \geq 0$, $v_n^i = A_i^{r+1}[s+1, t+1]$, where $r = \lfloor n/k^2 \rfloor$, $s = \lfloor (n \mod k^2)/k \rfloor$ and $t = n \mod k$. Therefore, $a_n^\star$ is non-negative iff $\sum_{i=1}^m w_i A_i^{r+1}[s+1, t+1]$ is non-negative. It follows that $\langle a^\star \rangle$ is ultimately non-negative iff $\sum_{i=1}^m w_i A_i^n$ is eventually non-negative. □

From Lemma 3, we can conclude the main result of this section, i.e., proof of Theorem 3. The following corollary can be shown *mutatis mutandis.*

Corollary 1. $\mathsf{EP}_{\mathsf{SoM}}$ *reduces to* $\mathsf{UP}_{\mathsf{LRS}}$ *in polynomial time.*

We note that it is also possible to argue about the eventual non-negativity (positivity) of only certain indices of the matrix using a similar argument as above. By interleaving only the LRS's corresponding to certain indices of the matrices in $\mathfrak{A}$, we can show this problem's equivalence with $\mathsf{UNN}_{\mathsf{LRS}}$ ($\mathsf{UP}_{\mathsf{LRS}}$).

5 Decision Procedures for Special Cases

Since there are no known algorithms for solving $\mathsf{UNN}_{\mathsf{LRS}}$ in general, the results of the previous section present a bleak picture for deciding $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$. We now show that these problems can be solved in some important special cases.

5.1 Simple Matrices and Matrices with Real Algebraic Eigenvalues

Our first positive result follows from known results for special classes of LRSes.

Theorem 4. $\mathsf{ENN}_{\mathsf{SoM}}$ *and* $\mathsf{EP}_{\mathsf{SoM}}$ *are decidable for* $\mathfrak{A} = \{(w_1, A_1), \ldots (w_m, A_m)\}$ *if one of the following conditions holds for all* $i \in \{1, \ldots m\}$.

1. *All A_i are simple. In this case,* $\mathsf{ENN}_{\mathsf{SoM}}$ *and* $\mathsf{EP}_{\mathsf{SoM}}$ *are in* PSPACE*. Additionally, if the rank k of all A_i is fixed,* $\mathsf{ENN}_{\mathsf{SoM}}$ *and* $\mathsf{EP}_{\mathsf{SoM}}$ *are in* PTIME*.*
2. *All eigenvalues of A_i are roots of real algebraic numbers. In this case,* $\mathsf{ENN}_{\mathsf{SoM}}$ *and* $\mathsf{EP}_{\mathsf{SoM}}$ *are in* $\mathsf{coNP}^{\mathsf{PosSLP}}$ *(a complexity class in the Counting Hierarchy, contained in* PSPACE*).*

Proof. Suppose each $A_i \in \mathbb{Q}^{k \times k}$, and let $\lambda_{i,1}, \ldots \lambda_{i,k}$ be the (possibly repeated) eigenvalues of A_i. The characteristic polynomial of A_i is $p_{A_i}(x) = \prod_{j=1}^k (x - \lambda_{i,j})$. Denote the LRS obtained from A_i by LRS interleaving as in Lemma 2 as $\langle a^i \rangle$. By Lemma 2, we have (i) $a_{rk^2+sk+t}^i = A_i^{r+1}[s+1, t+1]$ for all $r \in \mathbb{N}$ and $0 \leq s, t < k$, and (ii) $p_{\langle a^i \rangle}(x) = \prod_{j=1}^k \left(x^{k^2} - \lambda_{i,j}\right)$. We now define the scaled LRS $\{\langle b^i \rangle$, where $\mid b^i_n = w_i \; a^i_n$ for all $n \in \mathbb{N}$. Since scaling does not change the characteristic polynomial of an LRS (refer [3] for a simple proof), we have $p_{\langle b^i \rangle}(x) = \prod_{j=1}^{k} \left(x^{k^2} - \lambda_{i,j}\right)$. Once the LRSes $\langle b^1 \rangle, \ldots \langle b^m \rangle$ are obtained as above, we sum them to obtain the LRS $\langle b^\star \rangle$. Thus, for all $n \in \mathbb{N}$, we have $b^\star_n = \sum_{i=1}^{m} b^i_n = \sum_{i=1}^{m} w_i \; a^i_n = \sum_{i=1}^{m} w_i \; A_i^r[s,t]$, where $n = rk^2 + sk + t$, $r \in \mathbb{N}$ and $0 \le s, t < k$. Hence, $\mathsf{ENN}_{\mathsf{SoM}}$ (resp. $\mathsf{EP}_{\mathsf{SoM}}$) for $\{(w_1, A_1), \ldots (w_m, A_m)\}$ polynomially reduces to $\mathsf{UNN}_{\mathsf{LRS}}$ (resp. $\mathsf{UP}_{\mathsf{LRS}}$) for $\langle b^\star \rangle$.

By [14], we know that the characteristic polynomial $p_{\langle b^\star \rangle}(x)$ is the LCM of the characteristic polynomials $p_{\langle b^i \rangle}(x)$ for $1 \le i \le m$. If A_i are simple, there are no repeated roots of $p_{\langle b^i \rangle}(x)$. If this holds for all $i \in \{1, \ldots m\}$, there are no repeated roots of the LCM of $p_{\langle b^1 \rangle}(x), \ldots p_{\langle b^m \rangle}(x)$ as well. Hence, $p_{\langle b^\star \rangle}(x)$ has no repeated roots. Similarly, if all eigenvalues of A_i are roots of real algebraic numbers, so are all roots of $p_{\langle b^i \rangle}(x)$. It follows that all roots of the LCM of $p_{\langle b^1 \rangle}(x), \ldots p_{\langle b^m \rangle}(x)$, i.e. $p_{\langle b^\star \rangle}(x)$, are also roots of real algebraic numbers.

The theorem now follows from the following two known results about LRS.

1. $\mathsf{UNN}_{\mathsf{LRS}}$ (resp. $\mathsf{UP}_{\mathsf{LRS}}$) for simple LRS is in PSPACE. Furthermore, if the LRS is of bounded order, $\mathsf{UNN}_{\mathsf{LRS}}$ (resp. $\mathsf{UP}_{\mathsf{LRS}}$) is in PTIME [31].
2. $\mathsf{UNN}_{\mathsf{LRS}}$ (resp. $\mathsf{UP}_{\mathsf{LRS}}$) for LRS in which all roots of characteristic polynomial are roots of real algebraic numbers is in $\mathsf{coNP}^{\mathsf{PosSLP}}$ [2]. □

Remark: The technique used in [31] to decide $\mathsf{UNN}_{\mathsf{LRS}}$ (resp. $\mathsf{UP}_{\mathsf{LRS}}$) for simple rational LRS also works for simple LRS with real algebraic coefficients and initial values. This allows us to generalize Theorem 4(1) to the case where all A_i's and w_i's are real algebraic matrices and weights respectively.

5.2 Diagonalizable Matrices

We now ask if $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$ can be decided if each matrix A_i is diagonalizable. Since diagonalizable matrices strictly generalize simple matrices, Theorem 4(1) cannot answer this question directly, unless one perhaps looks under the hood of the (highly non-trivial) proof of decidability of non-negativity/positivity of simple LRSes. The main contribution of this section is a reduction that allows us to decide $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$ for diagonalizable matrices using a black-box decision procedure (i.e. without knowing operational details of the procedure or details of its proof of correctness) for the corresponding problem for simple real-algebraic matrices.

Before we proceed further, let us consider an example of a non-simple matrix (i.e. one with repeated eigenvalues) that is diagonalizable.

$$A = \begin{bmatrix} 5 & 12 & -6 \\ -3 & -10 & 6 \\ -3 & -12 & 8 \end{bmatrix}$$

Fig. 2. Diagonalizable matrix

Specifically, matrix A in Fig. 2 has eigenvalues $2, 2$ and -1, and can be written as SDS^{-1}, where D is the 3×3 diagonal matrix with $D[1,1] = D[2,2] = 2$ and $D[3,3] = -1$, and S is the 3×3 matrix with columns $(-4, 1, 0)^{\mathsf{T}}$, $(2, 0, 1)^{\mathsf{T}}$ and $(-1, 1, 1)^{\mathsf{T}}$.

Interestingly, the reduction technique we develop applies to properties much more general than $\mathsf{ENN}_{\mathsf{SoM}}$ and $\mathsf{EP}_{\mathsf{SoM}}$. Formally, given a sequence of matrices B_n defined by $\sum_{i=1}^{m} w_i A_i^n$, we say that a property $\mathcal{P}$ of the sequence is *positive scaling invariant* if it stays unchanged even if we scale all A_is by the same positive real. Examples of such properties include $\mathsf{ENN}_{\mathsf{SoM}}$, $\mathsf{EP}_{\mathsf{SoM}}$, non-negativity and positivity of B_n (i.e. is $B_n[i,j] \geq 0$ or < 0, as the case may be, for all $n \geq 1$ and for all $1 \leq i, j \leq k$), existence of zero (i.e. is B_n equal to the all 0-matrix for some $n \geq 1$), existence of a zero element (i.e. is $B_n[i,j] = 0$ for some $n \geq 1$ and some $i, j \in \{1, \ldots k\}$), variants of the r-non-negativity (resp. r-positivity and r-zero) problem, i.e. does there exist at least/exactly/at most r non-negative (resp. positive/zero) elements in B_n for all $n \geq 1$, for a given $r \in [1, k]$) etc. The main result of this section is a reduction for deciding such properties, formalized in the following theorem.

Theorem 5. *The decision problem for every positive scaling invariant property on rational diagonalizable matrices effectively reduces to the decision problem for the property on real algebraic simple matrices.*

While we defer the proof of this theorem to later in the section, an immediate consequence of Theorem 5 and Theorem 4(1) (read with the note at the end of Sect. 5.1) is the following result.

Corollary 2. $\mathsf{ENN}_{\mathsf{SoM}}$ *and* $\mathsf{EP}_{\mathsf{SoM}}$ *are decidable for* $\mathfrak{A} = \{(w_1, A_1), \ldots (w_m, A_m)\}$ *if all* A_i*s are rational diagonalizable matrices and all* w_i*s are rational.*

It is important to note that Theorem 5 yields a decision procedure for checking any positive scaling invariant property of diagonalizable matrices from a corresponding decision procedure for real algebraic simple matrices *without making any assumptions* about the inner working of the latter decision procedure. Given *any* black-box decision procedure for checking *any* positive scaling property for a set of weighted simple matrices, our reduction tells us how a corresponding decision procedure for checking the same property for a set of weighted diagonalizable matrices can be constructed. Interestingly, since diagonalizable matrices have an exponential form solution with constant coefficients for exponential terms, we can use an algorithm that exploits this specific property of the exponential form (like Ouaknine and Worrell's algorithm [31], originally proposed for checking ultimate positivity of simple LRS) to deal with diagonalizable matrices. However, our reduction technique is neither specific to this algorithm nor does it rely on any special property the exponential form of the solution.

The proof of Theorem 5 crucially relies on the notion of perturbation of diagonalizable matrices, which we introduce first. Let A be a $k \times k$ real diagonalizable matrix. Then, there exists an invertible $k \times k$ matrix S and a diagonal $k \times k$ matrix D such that $A = SDS^{-1}$, where S and D may have complex entries. It follows from basic linear algebra that for every $i \in \{1, \ldots k\}$, $D[i,i]$ is an eigenvalue of A and if α is an eigenvalue of A with algebraic multiplicity ρ, then α appears exactly ρ times along the diagonal of D. Furthermore, for every $i \in \{1, \ldots k\}$, the i^{th} column of S (resp. i^{th} row of S^{-1}) is an eigenvector of A (resp. of A^{T}) corresponding to the eigenvalue $D[i,i]$, and the columns of S

(resp. rows of S^{-1}) form a basis of the vector space $\mathbb{C}^k$. Let $\alpha_1, \dots \alpha_m$ be the eigenvalues of A with algebraic multiplicities $\rho_1, \dots \rho_m$ respectively. Wlog, we assume that $\rho_1 \geq \dots \geq \rho_m$ and the diagonal of D is partitioned into segments as follows: the first ρ_1 entries along the diagonal are α_1, the next ρ_2 entries are α_2, and so on. We refer to these segments as the α_1-segment, α_2-segment and so on, of diagonal of D. Formally, if κ_i denotes $\sum_{j=1}^{i-1} \rho_j$, the α_i-segment of diagonal of D consists of entries $D[\kappa_i + 1, \kappa_i + 1], \dots D[\kappa_i + \rho_i, \kappa_i + \rho_i]$, all of which are α_i.

Since A is a real matrix, its characteristic polynomial has all real coefficients and for every eigenvalue α of A (and hence of A^{T}), its complex conjugate, denoted $\overline{\alpha}$, is also an eigenvalue of A (and hence of A^{T}) with the same algebraic multiplicity. This allows us to define a bijection h_D from $\{1, \dots, k\}$ to $\{1, \dots k\}$ as follows. If $D[i, i]$ is real, then $h_D(i) = i$. Otherwise, let $D[i, i] = \alpha \in \mathbb{C}$ and let $D[i, i]$ be the l^{th} element in the α-segment of the diagonal of D. Then $h_D(i) = j$, where $D[j, j]$ is the l^{th} element in the $\overline{\alpha}$-segment of the diagonal of D. The matrix A being real also implies that for every real eigenvalue α of A (resp. of A^{T}), there exists a basis of *real eigenvectors* of the corresponding eigenspace. Additionally, for every non-real eigenvalue α and for every set of eigenvectors of A (resp. of A^{T}) that forms a basis of the eigenspace corresponding to α, the component-wise complex conjugates of these basis vectors serve as eigenvectors of A (resp. of A^{T}) and form a basis of the eigenspace corresponding to $\overline{\alpha}$.

Using the above notation, we choose matrix S^{-1} (and hence S) such that $A = SDS^{-1}$ as follows. Suppose α is an eigenvalue of A (and hence of A^{T}) with algebraic multiplicity ρ. Let $\{i + 1, \dots i + \rho\}$ be the set of indices j for which $D[j, j] = \alpha$. If α is real (resp. complex), the $i + 1^{st}, \dots i + \rho^{th}$ rows of S^{-1} are chosen to be real (resp. complex) eigenvectors of A^{T} that form a basis of the eigenspace corresponding to α. Moreover, if α is complex, the $h_D(i + s)^{th}$ row of S^{-1} is chosen to be the component-wise complex conjugate of the $i + s^{th}$ row of S^{-1}, for all $s \in \{1, \dots \rho\}$.

Definition 3. *Let $A = SDS^{-1}$ be a $k \times k$ real diagonalizable matrix. We say that $\mathcal{E} = (\varepsilon_1, \dots \varepsilon_k) \in \mathbb{R}^k$ is a* perturbation *w.r.t. D if $\varepsilon_i \neq 0$ and $\varepsilon_i = \varepsilon_{h_D(i)}$ for all $i \in \{1, \dots k\}$. Further, the $\mathcal{E}$*-perturbed variant of A *is the matrix $A' = SD'S^{-1}$, where D' is the $k \times k$ diagonal matrix with $D'[i, i] = \varepsilon_i D[i, i]$ for all $i \in \{1, \dots k\}$.*

In the following, we omit "w.r.t. D" and simply say "$\mathcal{E}$ is a perturbation", when D is clear from the context. Clearly, A' as defined above is a diagonalizable matrix and its eigenvalues are given by the diagonal elements of D'.

Recall that the diagonal of D is partitioned into α_i-segments, where each α_i is an eigenvalue of $A = SDS^{-1}$ with algebraic multiplicity ρ_i. We now use a similar idea to segment a perturbation $\mathcal{E}$ w.r.t. D. Specifically, the first ρ_1 elements of $\mathcal{E}$ constitute the α_1-segment of $\mathcal{E}$, the next ρ_2 elements of $\mathcal{E}$ constitute the α_2-segment of $\mathcal{E}$ and so on.

Definition 4. *A perturbation $\mathcal{E} = (\varepsilon_1, \dots \varepsilon_k)$ is said to be* segmented *if the j^{th} element (whenever present) in every segment of $\mathcal{E}$ has the same value, for all $1 \leq j \leq \rho_1$. Formally, if $i = \sum_{s=1}^{l-1} \rho_s + j$ and $1 \leq j \leq \rho_l \leq \rho_1$, then $\varepsilon_i = \varepsilon_j$.*

Clearly, the first ρ_1 elements of a segmented perturbation $\mathcal{E}$ define the whole of $\mathcal{E}$. As an example, suppose $(\alpha_1, \alpha_1, \alpha_1, \alpha_2, \alpha_2, \overline{\alpha_2}, \overline{\alpha_2}, \alpha_3)$ is the diagonal of D, where $\alpha_1, \alpha_2, \overline{\alpha_2}$ and α_3 are distinct eigenvalues of A. There are four segments of the diagonal of D (and of $\mathcal{E}$) of lengths $3, 2, 2$ and 1 respectively.

Example segmented perturbations in this case are $(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_1, \varepsilon_2, \varepsilon_1, \varepsilon_2, \varepsilon_1)$ and $(\varepsilon_3, \varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_1, \varepsilon_3, \varepsilon_1, \varepsilon_3)$. If $\varepsilon_1 \neq \varepsilon_2$ or $\varepsilon_2 \neq \varepsilon_3$, a perturbation that is *not segmented* is $\widetilde{\mathcal{E}} = (\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_2, \varepsilon_3, \varepsilon_2, \varepsilon_3, \varepsilon_1)$.

Definition 5. *Given a segmented perturbation $\mathcal{E} = (\varepsilon_1, \ldots \varepsilon_k)$ w.r.t. D, a* rotation *of $\mathcal{E}$, denoted $\tau_D(\mathcal{E})$, is the segmented perturbation $\mathcal{E}' = (\varepsilon'_1, \ldots \varepsilon'_k)$ in which $\varepsilon'_{(i \bmod \rho_1)+1} = \varepsilon_i$ for $i \in \{1, \ldots \rho_1\}$, and all other ε'_is are as in Definition 4.*

Continuing with our example, if $\mathcal{E} = (\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_1, \varepsilon_2, \varepsilon_1, \varepsilon_2, \varepsilon_1)$, then $\tau_D(\mathcal{E}) = (\varepsilon_3, \varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_1, \varepsilon_3, \varepsilon_1, \varepsilon_3)$, $\tau_D^2(\mathcal{E}) = (\varepsilon_2, \varepsilon_3, \varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_2, \varepsilon_3, \varepsilon_2)$ and $\tau_D^3(\mathcal{E}) = \mathcal{E}$.

Lemma 4. *Let $A = SDS^{-1}$ be a $k \times k$ real diagonalizable matrix with eigenvalues α_i of algebraic multiplicity ρ_i. Let $\mathcal{E} = (\varepsilon_1, \ldots \varepsilon_k)$ be a segmented perturbation w.r.t. D such that all ε_js have the same sign, and let A_u denote the $\tau_D^u(\mathcal{E})$-perturbed variant of A for $0 \leq u < \rho_1$, where $\tau^0(\mathcal{E}) = \mathcal{E}$. Then $A^n = \frac{1}{\left(\sum_{j=1}^{\rho_1} \varepsilon_j^n\right)} \sum_{u=0}^{\rho_1-1} A_u^n$, for all $n \geq 1$.*

Proof. Let $\mathcal{E}_u$ denote $\tau_D^u(\mathcal{E})$ for $0 \leq u < \rho_1$, and let $\mathcal{E}_u[i]$ denote the i^{th} element of $\mathcal{E}_u$ for $1 \leq i \leq k$. It follows from Definitions 4 and 5 that for each $i, j \in \{1, \ldots \rho_1\}$, there is a unique $u \in \{0, \ldots \rho_1 - 1\}$ such that $\mathcal{E}_u[i] = \varepsilon_j$. Specifically, $u = i - j$ if $i \geq j$, and $u = (\rho_1 - j) + i$ if $i < j$. Furthermore, Definition 4 ensures that the above property holds not only for $i \in \{1, \ldots \rho_1\}$, but for all $i \in \{1, \ldots k\}$.

Let D_u denote the diagonal matrix with $D_u[i,i] = \mathcal{E}_u[i] D[i,i]$ for $0 \leq i < \rho_1$. Then D_u^n is the diagonal matrix with $D_u^n[i,i] = \left(\mathcal{E}_u[i] D[i,i]\right)^n$ for all $n \geq 1$. It follows from the definition of A_u that $A_u^n = S\ D_u^n\ S^{-1}$ for $0 \leq u < \rho$ and $n \geq 1$. Therefore, $\sum_{u=0}^{\rho_1-1} A_u^n = S\ \left(\sum_{u=0}^{\rho_1-1} D_u^n\right)\ S^{-1}$. Now, $\sum_{u=0}^{\rho_1-1} D_u^n$ is a diagonal matrix whose i^{th} element along the diagonal is $\sum_{u=0}^{\rho_1-1} \left(\mathcal{E}_u[i] D[i,i]\right)^n$ $= \left(\sum_{u=0}^{\rho_1-1} \mathcal{E}_u^n[i]\right) D^n[i,i]$. By virtue of the property mentioned in the previous paragraph, $\sum_{u=0}^{\rho_1-1} \mathcal{E}_u^n[i] = \sum_{j=1}^{\rho_1} \varepsilon_j^n$ for $1 \leq i \leq k$. Therefore, $\sum_{u=0}^{\rho_1-1} D_u^n = \left(\sum_{j=1}^{\rho_1} \varepsilon_j^n\right) D^n$, and hence, $\sum_{u=0}^{\rho_1-1} A_u^n = \left(\sum_{j=1}^{\rho_1} \varepsilon_j^n\right) S\ D^n\ S^{-1} = \left(\sum_{j=1}^{\rho_1} \varepsilon_j^n\right) A^n$. Since all ε_js have the same sign and are non-zero, $\left(\sum_{j=1}^{\rho_1} \varepsilon_j^n\right)$ is non-zero for all $n \geq 1$. It follows that $A^n = \frac{1}{\left(\sum_{j=1}^{\rho_1} \varepsilon_j^n\right)} \sum_{u=0}^{\rho_1-1} A_u^n$. □

We are now in a position to present the proof of the main result of this section, i.e. of Theorem 5. Our proof uses a variation of the idea used in the proof of Lemma 4 above.

Proof of Theorem 5. Consider a set $\{(w_1, A_1), \ldots (w_i, A_i)\}$ of (weight, matrix) pairs, where each matrix A_i is in $\mathbb{Q}^{k \times k}$ and each $w_i \in \mathbb{Q}$. Suppose further that each $A_i = S_i D_i S_i^{-1}$, where D_i is a diagonal matrix with segments along the diagonal arranged in descending order of algebraic multiplicities of the corresponding eigenvalues. Let ν_i be the number of distinct eigenvalues of A_i, and let these eigenvalues be $\alpha_{i,1}, \ldots \alpha_{i,\nu_i}$. Let μ_i be the largest algebraic multiplicity among those of all eigenvalues of A_i, and let $\mu = lcm(\mu_1, \ldots \mu_m)$. We now choose *positive* rationals $\varepsilon_1, \ldots \varepsilon_\mu$ such that (i) all ε_js are distinct, and (ii) for every $i \in \{1, \ldots m\}$, for every distinct $j, l \in \{1, \ldots \nu_i\}$ and for every distinct $p, q \in \{1, \ldots \mu\}$, we have $\frac{\varepsilon_p}{\varepsilon_q} \neq |\frac{\alpha_{i,j}}{\alpha i,l}|$. Since $\mathbb{Q}$ is a dense set, such a choice of $\varepsilon_1, \ldots \varepsilon_\mu$ can always be made once all $|\frac{\alpha_{i,j}}{\alpha_{i,l}}|$s are known, even if within finite precision bounds.

For $1 \leq i \leq m$, let η_i denote μ/μ_i. We now define η_i distinct and segmented perturbations w.r.t. D_i as follows, and denote these as $\mathcal{E}_{i,1}, \ldots \mathcal{E}_{i,\eta_i}$. For $1 \leq j \leq \eta_i$, the first μ_i elements (i.e. the first segment) of $\mathcal{E}_{i,j}$ are $\varepsilon_{(j-1)\mu_i+1}, \ldots \varepsilon_{j\mu_i}$ (as chosen in the previous paragraph), and all other elements of $\mathcal{E}_{i,j}$ are defined as in Definition 4. For each $\mathcal{E}_{i,j}$ thus obtained, we also consider its rotations $\tau^u_{D_i}(\mathcal{E}_{i,j})$ for $0 \leq u < \mu_i$. For $1 \leq j \leq \eta_i$ and $0 \leq u < \mu_i$, let $A_{i,j,u} = S_i\ D_{i,j,u}\ S_i^{-1}$ denote the $\tau^u_{D_i}(\mathcal{E}_{i,j})$-perturbed variant of A_i. It follows from Definition 3 that if we consider the set of diagonal matrices $\{D_{i,j,u} \mid 1 \leq j \leq \eta_i,\ 0 \leq u < \mu_i\}$, then for every $p \in \{1, \ldots k\}$ and for every $q \in \{1, \ldots \mu\}$, there is a unique u and j such that $D_{i,j,u}[p,p] = \varepsilon_q$. Specifically, $j = \lfloor q/\mu_i \rfloor$. To find u, let $\mathcal{E}_{i,j}[p]$ be the $\widehat{p}^{th}$ element in a segment of $\mathcal{E}_{i,j}$, where $1 \leq \widehat{p} \leq \mu_i$, and let $\widehat{q}$ be $q \mod \mu_i$. Then, $u = (\widehat{p} - \widehat{q})$ if $\widehat{p} \geq \widehat{q}$ and $u = (\mu_i - \widehat{q}) + \widehat{p}$ otherwise. By our choice of ε_ts, we also know that for all $i \in \{1, \ldots m\}$, for all $j, l \in \{1, \ldots \nu_i\}$ and for all $p, q \in \{1, \ldots \mu\}$, we have $\varepsilon_p \alpha_{i,l} \neq \varepsilon_q \alpha_{i,j}$ unless $p = q$ *and* $j = l$. This ensures that all $D_{i,j,u}$ matrices, and hence all $A_{i,j,u}$s matrices, are simple, i.e. have distinct eigenvalues.

Using the reasoning in Lemma 4, we can now show that $A_i^n = \frac{1}{\left(\sum_{j=1}^{\mu} \varepsilon_j^n\right)} \times \left(\sum_{j=1}^{\eta_i} \sum_{u=0}^{\mu_i-1} A_{i,j,u}^n\right)$ and so, $\sum_{i=1}^m w_i A_i^n = \frac{1}{\left(\sum_{j=1}^{\mu} \varepsilon_j^n\right)} \times \left(\sum_{i=1}^m \sum_{j=1}^{\eta_i} \sum_{u=0}^{\mu_i-1} w_i A_{i,j,u}^n\right)$. Since all ε_js are positive reals, $\sum_{j=1}^{\mu} \varepsilon_j^n$ is a positive real for all $n \geq 1$.

Hence, for each $p, q \in \{1, \ldots k\}$, $\sum_{i=1}^m w_i A_i^n[p,q]$ is > 0, < 0 or $= 0$ if and only if $\left(\sum_{i=1}^m \sum_{j=1}^{\eta_i} \sum_{u=0}^{\mu_i-1} w_i A_{i,j,u}^n[p,q]\right)$ is > 0, < 0 or $= 0$, respectively. The only remaining helper result that is now needed to complete the proof of the theorem is that each $A_{i,j,u}$ is a real algebraic matrix. This is shown in Lemma 5, presented at the end of this section to minimally disturb the flow of arguments. □

The reduction in proof of Theorem 5 can be easily encoded as an algorithm, as shown in Algorithm 1. Further, in addition to Corollary 2, there are other consequences of our reduction. One such result (with proof in [3]) is below.

Corollary 3. *Given* $\mathfrak{A} = \{(w_1, A_1), \ldots (w_m, A_m)\}$, *where each* $w_i \in \mathbb{Q}$ *and* $A_i \in \mathbb{Q}^{k\times k}$ *is diagonalizable, and a real value* $\varepsilon > 0$, *there exists* $\mathfrak{B} = \{(v_1, B_1), \ldots (v_M, B_M)\}$, *where each* $v_i \in \mathbb{Q}$ *and each* $B_i \in \mathbb{R}\mathbb{A}^{k\times k}$ *is simple, such that* $\left|\sum_{i=0}^m w_i A_i^n[p,q] - \sum_{j=0}^M v_j B_j^n[p,q]\right| < \varepsilon^n$ *for all* $p, q \in \{1, \ldots k\}$ *and all* $n \geq 1$.

We end this section with the promised helper result used at the end of the proof of Theorem 5.

Algorithm 1. Reduction procedure for diagonalizable matrices

```
Input: $\mathfrak{A} = \{(w_i, A_i) \ : \ 1 \leq i \leq m, \ w_i \in \mathbb{Q}, \ A_i \in \mathbb{Q}^{k \times k}$ and diagonalizable$\}$
Output: $\mathfrak{B} = \{(v_i, B_i) : \ 1 \leq i \leq t, \ v_i \in \mathbb{Q}, \ B_i \in \mathbb{RA}^{k \times k}$ are simple$\}$
        s.t. $\left(\sum_{i=1}^{m} w_i A_i^n\right) = f(n) \left(\sum_{i=1}^{t} v_i B_i^n\right)$, where $f(n) > 0$ for all $n \geq 0$?
$P \leftarrow \{1\}$;                                   ▷ Initialize set of forbidden ratios of various $\varepsilon_j$s
for $i$ in 1 through $m$ do                                                    ▷ For each matrix $A_i$
    $R_i \leftarrow \{(\alpha_{i,j}, \rho_{i,j}) \ : \ \alpha_{i,j}$ is eigenvalue of $A_i$ with algebraic multiplicity $\rho_{i,j}\}$;
    $D_i \leftarrow$ Diagonal matrix of $\alpha_{i,j}$-segments ordered in decreasing order of $\rho_{i,j}$;
    $S_i \leftarrow$ Matrix of linearly independent eigenvectors of $A_i$ s.t. $A_i = S_i D_i S_i^{-1}$;
    $P \leftarrow P \cup \left\{ |\alpha_{i,j}/\alpha_{i,l}| \ : \ \alpha_{i,j}, \alpha_{i,l}\right.$ are eigenvalues in $\left.R_i\right\}$;   $\mu_i \leftarrow \max_j \rho_{i,j}$
$\mu = lcm(\mu_1, \ldots \mu_m)$;                                               ▷ Count of $\varepsilon_j$s needed
for $j$ in 1 through $\mu$ do                                         ▷ Generate all required $\varepsilon_j$s
    Choose $\varepsilon_j \in \mathbb{Q}$ s.t. $\varepsilon_j > 0$ and $\varepsilon_j \notin \{\pi \varepsilon_p \ : \ 1 \leq p < j, \ \pi \in P\}$;
$\mathfrak{B} \leftarrow \emptyset$;                               ▷ Initialize set of (weight, simple matrix) pairs
for $i$ in 1 through $m$ do                                                    ▷ For each matrix $A_i$
    $\nu_i \leftarrow \mu/\mu_i$;               ▷ Count of segmented perturbations to be rotated for $A_i$
    for $j$ in 0 through $\nu_i - 1$ do                          ▷ For each segmented perturbation
        $\mathcal{E}_{i,j}$ ← Seg. perturbn. w.r.t. $D_i$ with first $\mu_i$ elements being $\varepsilon_{j\mu_i+1}, \ldots \varepsilon_{(j+1)\mu_i}$;
        for $u$ in 0 through $\mu_i - 1$ do                          ▷ For each rotation of $\mathcal{E}_{i,j}$
            $A_{i,j,u} \leftarrow \tau_{D_i}^{u}(\mathcal{E}_{i,j})$-perturbed variant of $A$;
            $\mathfrak{B} \leftarrow \mathfrak{B} \cup \{(w_i, A_{i,j,u})\}$;                                  ▷ Update $\mathfrak{A}'$
return $\mathfrak{B}$;
```

Lemma 5. *For every real (resp. real algebraic) diagonalizable matrix $A = SDS^{-1}$ and perturbation $\mathcal{E} \in \mathbb{R}^k$ (resp. $\mathbb{RA}^k$), the $\mathcal{E}$-perturbed variant of A is a real (resp. real algebraic) diagonalizable matrix.*

Proof. We first consider the case of $A \in \mathbb{R}^{k \times k}$ and $\mathcal{E} \in \mathbb{R}^k$. Given a perturbation $\mathcal{E}$ w.r.t. D, we first define k *simple* perturbations $\mathcal{E}_i$ $(1 \leq i \leq k)$ w.r.t. D as follows: $\mathcal{E}_i$ has all its components set to 1, except for the i^{th} component, which is set to ε_i. Furthermore, if $D[i,i]$ is not real, then the $h_D(i)^{th}$ component of $\mathcal{E}_i$ is also set to ε_i. It is easy to see from Definition 3 that each $\mathcal{E}_i$ is a perturbation w.r.t. D. Moreover, if $j = h_D(i)$, then $\mathcal{E}_j = \mathcal{E}_i$.

Let $\widehat{\mathcal{E}} = \{\mathcal{E}_{i_1}, \ldots \mathcal{E}_{i_u}\}$ be the set of all *unique* perturbations w.r.t D among $\mathcal{E}_1, \ldots \mathcal{E}_k$. It follows once again from Definition 3 that the $\mathcal{E}$-perturbed variant of A can be obtained by a sequence of $\mathcal{E}_{i_j}$-perturbations, where $\mathcal{E}_{i_j} \in \widehat{\mathcal{E}}$. Specifically, let $A_{0,\widehat{\mathcal{E}}} = A$ and $A_{v,\widehat{\mathcal{E}}}$ be the $\mathcal{E}_{i_v}$-perturbed variant of $A_{v-1,\widehat{\mathcal{E}}}$ for all $v \in \{1, \ldots u\}$. Then, the $\mathcal{E}$-perturbed variant of A is identical to $A_{u,\widehat{\mathcal{E}}}$. This shows that it suffices to prove the lemma only for simple perturbations $\mathcal{E}_i$, as defined above. We focus on this special case below.

Let $A' = SD'S^{-1}$ be the $\mathcal{E}_i$-perturbed variant of A, and let $D[i,i] = \alpha$. For every $p \in \{1, \ldots k\}$, let $\mathbf{e_p}$ denote the p-dimensional unit vector whose p^{th} component is 1. Then, $A'\mathbf{e_p}$ gives the p^{th} column of A'. We prove the first part of the lemma by showing that $A' \ \mathbf{e_p} = (S \ D\ 'S^{-1}) \ \mathbf{e_p} \in \mathbb{R}^{k \times 1}$ for all $p \in \{1, \ldots k\}$.

Let $\mathbf{T}$ denote $D'\ S^{-1}\ \mathbf{e_p}$. Then $\mathbf{T}$ is a column vector with $\mathbf{T}[r] = D'[r,r]\ S^{-1}[r,p]$ for all $r \in \{1,\dots k\}$. Let $\mathbf{U}$ denote $S\mathbf{T}$. By definition, $\mathbf{U}$ is the p^{th} column of the matrix A'. To compute $\mathbf{U}$, recall that the rows of S^{-1} form a basis of $\mathbb{C}^k$. Therefore, for every $q \in \{1,\dots k\}$, $S^{-1}\ \mathbf{e_q}$ can be viewed as transforming the basis of the unit vector $\mathbf{e_q}$ to that given by the rows of S^{-1} (modulo possible scaling by real scalars denoting the lengths of the row vectors of S^{-1}). Similarly, computation of $\mathbf{U} = S\mathbf{T}$ can be viewed as applying the inverse basis transformation to $\mathbf{T}$. It follows that the components of $\mathbf{U}$ can be obtained by computing the dot product of $\mathbf{T}$ and the transformed unit vector $S^{-1}\ \mathbf{e_q}$, for each $q \in \{1,\dots k\}$. In other words, $\mathbf{U}[q] = \mathbf{T} \cdot (S^{-1}\ \mathbf{e_q})$. We show below that each such $\mathbf{U}[q]$ is real.

By definition, $\mathbf{U}[q] = \sum_{r=1}^{k}(\mathbf{T}[r]\ S^{-1}[r,q]) = \sum_{r=1}^{k}(D'[r,r]\ S^{-1}[r,p]\ S^{-1}[r,q])$. We consider two cases below.

- If $D[i,i] = \alpha$ is real, recalling the definition of D', the expression for $\mathbf{U}[q]$ simplifies to $\sum_{r=1}^{k}(D[r,r]\ S^{-1}[r,p]\ S^{-1}[r,q]) + (\varepsilon_i - 1)\ \alpha\ S^{-1}[i,p]\ S^{-1}[i,q]$. Note that $\sum_{r=1}^{k}(D[r,r]\ S^{-1}[r,p]\ S^{-1}[r,q])$ is the q^{th} component of the vector $(SDS^{-1})\ \mathbf{e_p} = A\ \mathbf{e_p}$. Since A is real, so must be the q^{th} component of $A\ \mathbf{e_p}$. Moreover, since α is real, by our choice of S^{-1}, both $S^{-1}[i,p]$ and $S^{-1}[i,q]$ are real. Since ε_i is also real, it follows that $(\varepsilon_i - 1)\ \alpha\ S^{-1}[i,p]\ S^{-1}[i,q]$ is real. Hence $\mathbf{U}[q]$ is real for all $q \in \{1,\dots k\}$.
- If $D[i,i] = \alpha$ is not real, from Definition 3, we know that $D'[i,i] = \varepsilon_i\ \alpha$ and $D'[h_D(i), h_D(i)] = \varepsilon_i\ \overline{\alpha}$. The expression for $\mathbf{U}[q]$ then simplifies to $\sum_{r=1}^{k}\big(D[r,r]\ S^{-1}[r,p]\ S^{-1}[r,q]\big) + (\varepsilon_i - 1)\ (\beta + \gamma)$, where $\beta = \alpha\ S^{-1}[i,p]\ S^{-1}[i,q]$ and $\gamma = \overline{\alpha}\ S^{-1}[h_D(i),p]\ S^{-1}[h_D(i),q]$. By our choice of S^{-1}, we know that $S^{-1}[h_D(i),p] = \overline{S^{-1}[i,p]}$ and $S^{-1}[h_D(i),q] = \overline{S^{-1}[i,q]}$. Therefore, $\beta = \overline{\gamma}$ and hence $(\varepsilon_i - 1)\ (\beta + \gamma)$ is real. By a similar argument as in the previous case, it follows that $\mathbf{U}[q]$ is real for all $q \in \{1,\dots k\}$.

The proof when $A \in \mathbb{RA}^{k\times k}$ and $\mathcal{E} \in \mathbb{Q}^k$ follows from a similar reasoning as above, and from the following facts about real algebraic matrices.

- If A is a real algebraic matrix, then every eigenvalue of A is either a real or complex algebraic number.
- If A is diagonalizable, then for every real (resp. complex) algebraic eigenvalue of A, there exists a set of real (resp. complex) algebraic eigenvectors that form a basis of the corresponding eigenspace. □

6 Conclusion

In this paper, we investigated eventual non-negativity and positivity for matrices and the weighted sum of powers of matrices ($\mathsf{ENN_{SoM}}/\mathsf{EP_{SoM}}$). First, we showed reductions from and to specific problems on linear recurrences, which allowed us give complexity lower and upper bounds. Second, we developed a new and generic perturbation-based reduction technique from simple matrices to diagonalizable matrices, which allowed us to transfer results between these settings.

Most of our results, that we showed in the rational setting, hold even with real-algebraic matrices by adapting the complexity notions and depending on corresponding results for ultimate positivity for linear recurrences and related problems over reals. As future work, we would like to extend our techniques for other problems of interest like the *existence* of a matrix power where all entries are non-negative or zero. Finally, the line of work started here could lead to effective algorithms and applications in varied areas ranging from control theory systems to cyber-physical systems, where eventual properties of matrices play a crucial role.

References

1. Akshay, S., Antonopoulos, T., Ouaknine, J., Worrell, J.: Reachability problems for Markov chains. Inf. Process. Lett. **115**(2), 155–158 (2015)
2. Akshay, S., Balaji, N., Murhekar, A., Varma, R., Vyas, N.: Near optimal complexity bounds for fragments of the Skolem problem. In: Paul, S., Bläser, M. (eds.) 37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, 10–13 March 2020, Montpellier, France, volume 154 of LIPIcs, pp. 37:1–37:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
3. Akshay, S., Chakraborty, S., Pal, D.: On eventual non-negativity and positivity for the weighted sum of powers of matrices. arXiv preprint arXiv:2205.09190 (2022)
4. Akshay, S., Genest, B., Karelovic, B., Vyas, N.: On regularity of unary probabilistic automata. In: 33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, 17–20 February 2016, Orléans, France, volume 47 of LIPIcs, pp. 8:1–8:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016)
5. S. Akshay, Blaise Genest, and Nikhil Vyas. Distribution based objectives for Markov decision processes. In: 33rd Symposium on Logic in Computer Science (LICS 2018), vol. IEEE, pp. 36–45 (2018)
6. Almagor, S., Boker, U., Kupferman, O.: What's decidable about weighted automata? Inf. Comput. **282**, 104651 (2020)
7. Almagor, S., Karimov, T., Kelmendi, E., Ouaknine, J., Worrell, J.: Deciding ω-regular properties on linear recurrence sequences. Proc. ACM Program. Lang. **5**(POPL), 1–24 (2021)
8. Barloy, C., Fijalkow, N., Lhote, N., Mazowiecki, F.: A robust class of linear recurrence sequences. In: Fernández, M., Muscholl, A. (eds.) 28th EACSL Annual Conference on Computer Science Logic, CSL 2020, 13–16 January 2020, Barcelona, Spain, volume 152 of LIPIcs, pp. 9:1–9:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
9. Bell, P.C., Hirvensalo, M., Potapov, I.: Mortality for 2 × 2 matrices is NP-hard. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 148–159. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32589-2_16
10. Bell, P.C., Potapov, I., Semukhin, P.: On the mortality problem: from multiplicative matrix equations to linear recurrence sequences and beyond. Inf. Comput. **281**, 104736 (2021)
11. Bell, P.C., Semukhin, P.: Decision questions for probabilistic automata on small alphabets. arXiv preprint arXiv:2105.10293 (2021)
12. Blondel, V.D., Canterini, V.: Undecidable problems for probabilistic automata of fixed dimension. Theory Comput. Syst. **36**(3), 231–245 (2003). https://doi.org/10.1007/s00224-003-1061-2

13. Naqvi, S.C., McDonald, J.J.: Eventually nonnegative matrices are similar to seminonnegative matrices. Linear Algebra Appl. **381**, 245–258 (2004)
14. Everest, G., van der Poorten, A., Shparlinski, I., Ward, T.: Recurrence Sequences. Mathematical Surveys and Monographs, American Mathematical Society, United States (2003)
15. Fijalkow, N., Ouaknine, J., Pouly, A., Sousa-Pinto, J., Worrell, J.: On the decidability of reachability in linear time-invariant systems. In: Ozay, N., Prabhakar, P. (eds.) Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, 16–18 April 2019, pages 77–86. ACM (2019)
16. Friedland, S.: On an inverse problem for nonnegative and eventually nonnegative matrices. Isr. J. Math. **29**(1), 43–60 (1978). https://doi.org/10.1007/BF02760401
17. Gimbert, H., Oualhadj, Y.: Probabilistic automata on finite words: decidable and undecidable problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 527–538. Springer, . Probabilistic automata on finite words: Decidable and undecidable problems (2010). https://doi.org/10.1007/978-3-642-14162-1_44
18. Halava, V., Harju, T., Hirvensalo, M.: Positivity of second order linear recurrent sequences. Discrete Appl. Math. **154**(3), 447–451 (2006)
19. Halava, V., Harju, T., Hirvensalo, M., Karhumäki, J.: Skolem's Problem-on the Border Between Decidability and Undecidability. Technical report, Citeseer (2005)
20. Karimov, T., et al.: What's decidable about linear loops? Proc. ACM Program. Lang. **6**(POPL), 1–25 (2022)
21. Korthikanti, V.A., Viswanathan, M., Agha, G., Kwon, Y.: Reasoning about MDPs as transformers of probability distributions. In: QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, Williamsburg, Virginia, USA, 15–18 September 2010, pp. 199–208. IEEE Computer Society (2010)
22. Lale, S., Azizzadenesheli, K., Hassibi, B., Anandkumar, A.: Logarithmic regret bound in partially observable linear dynamical systems. Adv. Neural Inf. Process. Syst. **33**, 20876–20888 (2020)
23. Lebacque, J.P., Ma, T.Y., Khoshyaran, M.M.: The cross-entropy field for multimodal dynamic assignment. In: Proceedings of Traffic and Granular Flow 2009 (2009)
24. MacCluer, C.R.: The many proofs and applications of Perron's theorem. Siam Rev. **42**(3), 487–498 (2000)
25. Noutsos, D.: On Perron-Frobenius property of matrices having some negative entries. Linear Algebra Appl. **412**(2), 132–153 (2006)
26. Noutsos, D., Tsatsomeros, M.J.: Reachability and holdability of nonnegative states. SIAM J. Matrix Anal. Appl. **30**(2), 700–712 (2008)
27. Ouaknine, J.: Decision problems for linear recurrence sequences. In: Gąsieniec, L., Wolter, F. (eds.) FCT 2013. LNCS, vol. 8070, pp. 2–2. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40164-0_2
28. Ouaknine, J., Pinto, J.S., Worrell, J.: On termination of integer linear loops. In: Indyk, R. (ed.) Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, 4–6 January 2015, pp. 957–969. SIAM (2015)
29. Ouaknine, J., Worrell, J.: Decision problems for linear recurrence sequences. In: Finkel, A., Leroux, J., Potapov, I. (eds.) RP 2012. LNCS, vol. 7550, pp. 21–28. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33512-9_3

30. Ouaknine, J., Worrell, J.: Positivity problems for low-order linear recurrence sequences. In: Chekuri, C. (ed.) Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, 5–7 January 2014, pp. 366–379. SIAM (2014)
31. Ouaknine, J., Worrell, J.: Ultimate positivity is decidable for simple linear recurrence sequences. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8573, pp. 330–341. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43951-7_28
32. Ouaknine, J., Worrell, J.: On linear recurrence sequences and loop termination. ACM Siglog News **2**(2), 4–13 (2015)
33. Pan, V.Y., Chen, Z.Q.: The complexity of the matrix eigenproblem. In: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, STOC 2099, pp. 507–516, New York, NY, USA. Association for Computing Machinery (1999)
34. Rump, S.M.: Perron-Frobenius theory for complex matrices. Linear Algebra Appl. **363**, 251–273 (2003)
35. Akshay, S., Balaji, N., Vyas, N.: Complexity of Restricted Variants of Skolem and Related Problems. In Larsen, K.G., Bodlaender, H.L., Raskin, J.F. (eds.) 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017), volume 83 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 78:1–78:14, Dagstuhl, Germany. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
36. Tiwari, A.: Termination of linear programs. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 70–82. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_6
37. Zaslavsky, B.G.: Eventually nonnegative realization of difference control systems. Dyn. Syst. Relat. Top. Adv. Ser. Dynam. Syst. **9**, 573–602 (1991)
38. Zaslavsky, B.G., McDonald, J.J.: Characterization of Jordan canonical forms which are similar to eventually nonnegative matrices with the properties of nonnegative matrices. Linear Algebra Appl. **372**, 253–285 (2003)
39. Zhang, A., et al.: Learning causal state representations of partially observable environments. arXiv preprint arXiv:1906.10437 (2019)

Decision Problems in a Logic for Reasoning About Reconfigurable Distributed Systems

Marius Bozga(✉), Lucas Bueri, and Radu Iosif

Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Saint-Martin-d'Hères, France
marius.bozga@univ-grenoble-alpes.fr

Abstract. We consider a logic used to describe sets of configurations of distributed systems, whose network topologies can be changed at runtime, by reconfiguration programs. The logic uses inductive definitions to describe networks with an unbounded number of components and interactions, written using a multiplicative conjunction, reminiscent of Bunched Implications [37] and Separation Logic [39]. We study the complexity of the satisfiability and entailment problems for the configuration logic under consideration. Additionally, we consider the robustness property of degree boundedness (is every component involved in a bounded number of interactions?), an ingredient for decidability of entailments.

1 Introduction

Distributed systems are increasingly used as critical parts of the infrastructure of our digital society, as in e.g., datacenters, e-banking and social networking. In order to address maintenance (e.g., replacement of faulty and obsolete network nodes by new ones) and data traffic issues (e.g., managing the traffic inside a datacenter [35]), the distributed systems community has recently put massive effort in designing algorithms for *reconfigurable systems*, whose network topologies change at runtime [23]. However, dynamic reconfiguration in the form of software or network upgrades has been recognized as one of the most important sources of cloud service outage [25].

This paper contributes to a logical framework that addresses the timely problems of formal *modeling* and *verification* of reconfigurable distributed systems. The basic building blocks of this framework are (i) a Hoare-style program proof calculus [1] used to write formal proofs of correctness of reconfiguration programs, and (ii) an invariant synthesis method [6] that proves the safety (i.e., absence of reachable error configurations) of the configurations defined by the assertions that annotate a reconfiguration program. These methods are combined to prove that an initially correct distributed system cannot reach an error state, following the execution of a given reconfiguration sequence.

The assertions of the proof calculus are written in a logic that defines infinite sets of configurations, consisting of *components* (i.e., processes running on different nodes of the network) connected by *interactions* (i.e., multi-party channels alongside which messages between components are transfered). Systems that share the same architectural style (e.g., pipeline, ring, star, tree, etc.) and differ by the number of components and interactions are described using inductively defined predicates. Such configurations can be modified either by (a) adding or removing components and interactions (reconfiguration), or (b) changing the local states of components, by firing interactions.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 691–711, 2022.
https://doi.org/10.1007/978-3-031-10769-6_40

The assertion logic views components and interactions as *resources*, that can be created or deleted, in the spirit of resource logics *à la* Bunched Implications [37], or Separation Logic [39]. The main advantage of using resource logics is their support for *local reasoning* [12]: reconfiguration actions are specified by pre- and postconditions mentioning only the resources involved, while framing out the rest of the configuration.

The price to pay for this expressive power is the difficulty of automating the reasoning in these logics. This paper makes several contributions in the direction of proof automation, by studying the complexity of the *satisfiability* and *entailment* problems, for the configuration logic under consideration. Additionally, we study the complexity of a robustness property [27], namely *degree boundedness* (is every component involved in a bounded number of interactions?). In particular, the latter problem is used as a prerequisite for defining a fragment with a decidable entailment problem. For space reasons, the proofs of the technical results are given in [5].

1.1 Motivating Example

The logic studied in this paper is motivated by the need for an assertion language that supports reasoning about dynamic reconfigurations in a distributed system. For instance, consider a distributed system consisting of a finite (but unknown) number of *components* (processes) placed in a ring, executing the same finite-state program and communicating via *interactions* that connect the *out* port of a component to the *in* port of its right neighbour, in a round-robin fashion, as in Fig. 1(a). The behavior of a component is a machine with two states, T and H, denoting whether the component has a token (T) or not (H). A component c_i without a token may receive one, by executing a transition $\mathsf{H} \xrightarrow{in} \mathsf{T}$, simultaneously with its left neighbour c_j, that executes the transition $\mathsf{T} \xrightarrow{out} \mathsf{H}$. Then, we say that the interaction (c_j, out, c_i, in) has fired, moving a token one position to the right in the ring. Note that there can be more than one token, moving independently in the system, as long as no token overtakes another token.

The token ring system is formally specified by the following inductive rules:

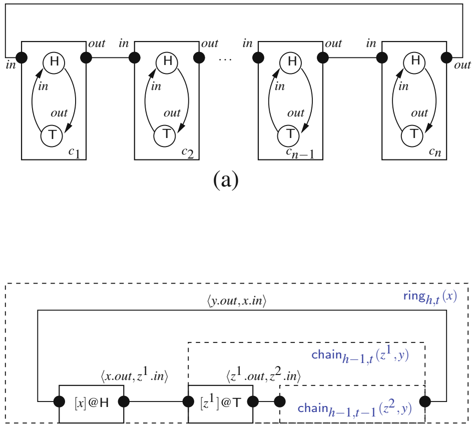
$$\mathsf{ring}_{h,t}(x) \leftarrow \exists y \exists z \,.\, [x]@q * \langle x.out, z.in\rangle * \mathsf{chain}_{h',t'}(z,y) * \langle y.out, x.in\rangle$$

$$\mathsf{chain}_{h,t}(x,y) \leftarrow \exists z.\ [x]@q * \langle x.out, z.in\rangle * \mathsf{chain}_{h',t'}(z,y)$$

$$\mathsf{chain}_{0,1}(x,x) \leftarrow [x]@\mathsf{T} \qquad \mathsf{chain}_{1,0}(x,x) \leftarrow [x]@\mathsf{H} \qquad \mathsf{chain}_{0,0}(x,x) \leftarrow [x]$$

$$\text{where } h' \overset{\text{def}}{=} \begin{cases} \max(h-1,0) & \text{, if } q = \mathsf{H} \\ h & \text{, if } q = \mathsf{T} \end{cases} \text{ and } t' \overset{\text{def}}{=} \begin{cases} \max(t-1,0) & \text{, if } q = \mathsf{T} \\ t & \text{, if } q = \mathsf{H} \end{cases}$$

The predicate $\mathsf{ring}_{h,t}(x)$ describes a ring with at least two components, such that at least h (resp. t) components are in state H (resp. T). The ring consists of a component x in state q, described by the formula $[x]@q$, an interaction from the *out* port of x to the *in* port of another component z, described as $\langle x.out, z.in\rangle$, a separate chain of components stretching from z to y ($\mathsf{chain}_{h',t'}(z,y)$), and an interaction connecting the *out* port of component y to the *in* port of component x ($\langle y.out, x.in\rangle$). Inductively, a chain consists of a component $[x]@q$, an interaction $\langle x.out, z.in\rangle$ and a separate $\mathsf{chain}_{h',t'}(z,y)$. Figure 1(b) depicts the unfolding of the inductive definition of the token ring, with the


in
out
in
out
in
out
in
out
H
T
in
out
c_1
c_2
$\cdots$
c_{n-1}
c_n
(a)
$\langle y.out, x.in\rangle$
$\mathsf{ring}_{h,t}(x)$
$\mathsf{chain}_{h-1,t}(z^1, y)$
$\langle x.out, z^1.in\rangle$
$\langle z^1.out, z^2.in\rangle$
$[x]$@H
$[z^1]$@T
$\mathsf{chain}_{h-1,t-1}(z^2, y)$

(b)

```
$\{\mathsf{ring}_{h,t}(y)\}$  /*assume $h \geq 2$, $t \geq 1$*/
$\left\{\begin{array}{l}\boxed{[y]@\mathsf{H} * \langle y.out, z.in\rangle * \mathsf{chain}_{h-1,t}(z,x)} \\ * \langle x.out, y.in\rangle\end{array}\right\}$
disconnect(x.$out$, y.$in$);
$\{\boxed{[y]@\mathsf{H}} * \langle y.out, z.in\rangle * \boxed{\mathsf{chain}_{h-t,t}(z,x)}\}$
disconnect(y.$out$, z.$in$);
$\{[y]@\mathsf{H} * \boxed{\mathsf{chain}_{h-1,t}(z,x)}\}$
delete(y);
$\{\boxed{\mathsf{chain}_{h-1,t}(z,x)}\}$
connect(x.$out$, z.$in$)
$\{\mathsf{chain}_{h-1,t}(z,x) * \langle x.out, z.in\rangle\}$
$\{\mathsf{ring}_{h-1,t}(z)\}$
```

(c)

Fig. 1. Inductive Specification and Reconfiguration of a Token Ring

existentially quantified variables z from the above rules α-renamed to $z^1, z^2, \ldots$ to avoid confusion.

A *reconfiguration program* takes as input a mapping of program variables to components and executes a sequence of *basic operations* i.e., component/interaction creation/deletion, involving the components and interactions denoted by these variables. For instance, the reconfiguration program in Fig. 1(c) takes as input three adjacent components, mapped to the variables x, y and z, respectively, removes the component y together with its left and right interactions and reconnects x directly with z. Programming reconfigurations is error-prone, because the interleaving between reconfiguration actions and interactions in a distributed system may lead to bugs that are hard to trace. For instance, if a reconfiguration program removes the last component in state T (resp. H) from the system, no token transfer interaction may fire and the system deadlocks.

We prove absence of such errors using a Hoare-style proof system [1], based on the logic introduced above as assertion language. For instance, the proof from Fig. 1(c) shows that the reconfiguration sequence applied to a component y in state H (i.e., $[y]$@H) in a ring with at least $h \geq 2$ components in state H and at least $t \geq 1$ components in state T leads to a ring with at least $h-1$ components in state H and at least t in state T; note that the states of the components may change during the execution of the reconfiguration program, as tokens are moved by interactions.

The proof in Fig. 1(c) uses *local axioms* specifying, for each basic operation, only those components and interactions required to avoid faulting, with a *frame rule* $\{\phi\}$ P $\{\psi\} \Rightarrow \{\phi * \boxed{F}\}$ P $\{\psi * F\}$; for readability, the frame formulæ (from the preconditions of the conclusion of the frame rule applications) are enclosed in boxes.

The proof also uses the *consequence rule* $\{\phi\}$ P $\{\psi\} \Rightarrow \{\phi'\}$ P $\{\psi'\}$ that applies if ϕ' is stronger than ϕ and ψ' is weaker than ψ. The side conditions of the consequence rule require checking the validity of the entailments $\mathsf{ring}_{h,t}(y) \models \exists x \exists z \,.\, \langle x.out, y.in\rangle * [y]@\mathsf{H} * \langle y.out, z.in\rangle * \mathsf{chain}_{h-1,t}(z,x)$ and $\mathsf{chain}_{h-1,t}(z,x) * \langle x.out, z.in\rangle \models \mathsf{ring}_{h-1,t}(z)$, for all $h \geq 2$ and $t \geq 1$. These side conditions can be automatically discharged using the results on the decidability of entailments given in this paper. Additionally, checking the satisfiability of a precondition is used to detect trivially valid Hoare triples.

1.2 Related Work

Formal modeling coordinating architectures of component-based systems has received lots of attention, with the development of architecture description languages (ADL), such as BIP [3] or REO [2]. Many such ADLs have extensions that describe programmed reconfiguration, e.g., [19,30], classified according to the underlying formalism used to define their operational semantics: *process algebras* [13,33], *graph rewriting* [32,41,44], *chemical reactions* [43] (see the surveys [7,11]). Unfortunately, only few ADLs support formal verification, mainly in the flavour of runtime verification [10,17,20,31] or finite-state model checking [14].

Parameterized verification of unbounded networks of distributed processes uses mostly hard-coded coordinating architectures (see [4] for a survey). A first attempt at specifying architectures by logic is the *interaction logic* of Konnov et al. [29], a combination of Presburger arithmetic with monadic uninterpreted function symbols, that can describe cliques, stars and rings. More structured architectures (pipelines and trees) can be described using a second-order extension [34]. However, these interaction logics are undecidable and lack support for automated reasoning.

Specifying parameterized component-based systems by inductive definitions is not new. *Network grammars* [26,32,40] use context-free grammar rules to describe systems with linear (pipeline, token-ring) architectures obtained by composition of an unbounded number of processes. In contrast, we use predicates of unrestricted arities to describe architectural styles that are, in general, more complex than trees. Moreover, we write inductive definitions using a resource logic, suitable also for writing Hoare logic proofs of reconfiguration programs, based on local reasoning [12].

Local reasoning about concurrent programs has been traditionally the focus of Concurrent Separation Logic (CSL), based on a parallel composition rule [36], initially with a non-interfering (race-free) semantics [8] and later combining ideas of assume- and rely-guarantee [28,38] with local reasoning [22,42] and abstract notions of framing [15,16,21]. However, the body of work on CSL deals almost entirely with shared-memory multithreading programs, instead of distributed systems, which is the aim of our work. In contrast, we develop a resource logic in which the processes do not just share and own resources, but become mutable resources themselves.

The techniques developed in this paper are inspired by existing techniques for similar problems in the context of Separation Logic (SL) [39]. For instance, we use an abstract domain similar to the one defined by Brotherston et al. [9] for checking satisfiability of symbolic heaps in SL and reduce a fragment of the entailment problem in our logic to SL entailment [18]. In particular, the use of existing automated reasoning techniques for SL has pointed out several differences between the expressiveness of our logic and that of SL. First, the configuration logic describes hypergraph structures, in which edges are ℓ-tuples for $\ell \geq 2$, instead of directed graphs as in SL, where ℓ is a parameter of the problem: considering ℓ to be a constant strictly decreases the complexity of the problem. Second, the degree (number of hyperedges containing a given vertex) is unbounded, unlike in SL, where the degree of heaps is constant. Therefore, we dedicate an entire section (Sect. 4) to the problem of deciding the existence of a bound (and computing a cut-off) on the degree of the models of a formula, used as a prerequisite for the encoding of the entailment problems from the configuration logic as SL entailments.

2 Definitions

We denote by $\mathbb{N}$ the set of positive integers including zero. For a set A, we define $A^1 \stackrel{\mathsf{def}}{=} A$, $A^{i+1} \stackrel{\mathsf{def}}{=} A^i \times A$, for all $i \geq 1$, and $A^+ = \bigcup_{i \geq 1} A^i$, where $\times$ denotes the Cartesian product. We denote by $\mathrm{pow}(A)$ the powerset of A and by $\mathrm{mpow}(A)$ the power-multiset (set of multisets) of A. The cardinality of a finite set A is denoted as $\|A\|$. By writing $A \subseteq_{\mathit{fin}} B$ we mean that A is a finite subset of B. Given integers i and j, we write $[i, j]$ for the set $\{i, i+1, \ldots, j\}$, assumed to be empty if $i > j$. For a tuple $\mathbf{t} = \langle t_1, \ldots, t_n \rangle$, we define $|\mathbf{t}| \stackrel{\mathsf{def}}{=} n$, $\langle \mathbf{t} \rangle_i \stackrel{\mathsf{def}}{=} t_i$ and $\langle \mathbf{t} \rangle_{[i,j]} \stackrel{\mathsf{def}}{=} \langle t_i, \ldots, t_j \rangle$. By writing $x = \mathit{poly}(y)$, for given $x, y \in \mathbb{N}$, we mean that there exists a polynomial function $f : \mathbb{N} \rightarrow \mathbb{N}$, such that $x \leq f(y)$.

2.1 Configurations

We model distributed systems as hypergraphs, whose vertices are *components* (i.e., the nodes of the network) and hyperedges are *interactions* (i.e., describing the way the components communicate with each other). The components are taken from a countably infinite set $\mathbb{C}$, called the *universe*. We consider that each component executes its own copy of the same *behavior*, represented as a finite-state machine $\mathbb{B} = (\mathcal{P}, \mathcal{Q}, \rightarrow)$, where $\mathcal{P}$ is a finite set of *ports*, $\mathcal{Q}$ is a finite set of *states* and $\rightarrow \subseteq \mathcal{Q} \times \mathcal{P} \times \mathcal{Q}$ is a transition relation. Intuitively, each transition $q \xrightarrow{p} q'$ of the behavior is triggerred by a visible event, represented by the port p. For instance, the behavior of the components of the token ring system from Fig. 1(a) is $\mathbb{B} = (\{\mathit{in}, \mathit{out}\}, \{\mathsf{H}, \mathsf{T}\}, \{\mathsf{H} \xrightarrow{\mathit{in}} \mathsf{T}, \mathsf{T} \xrightarrow{\mathit{out}} \mathsf{H}\})$. *The universe $\mathbb{C}$ and the behavior $\mathbb{B} = (\mathcal{P}, \mathcal{Q}, \rightarrow)$ are fixed in the rest of this paper.*

We introduce a logic for describing infinite sets of *configurations* of distributed systems with unboundedly many components and interactions. A configuration is a snapshot of the system, describing the topology of the network (i.e., the set of present components and interactions) together with the local state of each component:

Definition 1. *A* configuration *is a tuple $\gamma = (\mathcal{C}, \mathcal{I}, \rho)$, where:*

- *$\mathcal{C} \subseteq_{\mathit{fin}} \mathbb{C}$ is a finite set of* components*, that are present in the configuration,*
- *$\mathcal{I} \subseteq_{\mathit{fin}} (\mathbb{C} \times \mathcal{P})^+$ is a finite set of* interactions*, where each interaction is a sequence $(c_1, p_1, \ldots, c_n, p_n) \in (\mathbb{C} \times \mathcal{P})^n$ that binds together the ports $p_1, \ldots, p_n$ of the pairwise distinct components $c_1, \ldots, c_n$, respectively.*
- *$\rho : \mathbb{C} \rightarrow \mathcal{Q}$ is a* state map *associating each (possibly absent) component, a state of the behavior $\mathbb{B}$, such that the set $\{c \in \mathbb{C} \mid \rho(c) = q\}$ is infinite, for each $q \in \mathcal{Q}$.*

The last condition requires that there is an infinite pool of components in each state $q \in \mathcal{Q}$; since $\mathbb{C}$ is infinite and $\mathcal{Q}$ is finite, this condition is feasible. For example, the configurations of the token ring from Fig. 1(a) are $(\{c_1, \ldots, c_n\}, \{(c_i, \mathit{out}, c_{(i \bmod n)+1}, \mathit{in}) \mid$

$i \in [1,n]\}, \rho)$, where $\rho : \mathbb{C} \to \{\mathsf{H}, \mathsf{T}\}$ is a state map. The ring topology is described by the set of components $\{c_1, \ldots, c_n\}$ and interactions $\{(c_i, out, c_{(i \bmod n)+1}, in) \mid i \in [1,n]\}$.

Intuitively, an interaction $(c_1, p_1, \ldots, c_n, p_n)$ synchronizes transitions labeled by the ports $p_1, \ldots, p_n$ from the behaviors (i.e., replicas of the state machine $\mathbb{B}$) of $c_1, \ldots, c_n$, respectively. Note that the components c_i are not necessary part of the configuration. The interactions are classified according to their sequence of ports, called the *interaction type* and let $\mathsf{Inter} \stackrel{\mathsf{def}}{=} \mathcal{P}^+$ be the set of interaction types; an interaction type models, for instance, the passing of a certain kind of message (e.g., request, acknowledgement, etc.). From an operational point of view, two interactions that differ by a permutation of indices e.g., $(c_1, p_1, \ldots, c_n, p_n)$ and $(c_{i_1}, p_{i_1}, \ldots, c_{i_n}, p_{i_n})$ such that $\{i_1, \ldots, i_n\} = [1,n]$, are equivalent, since the set of transitions is the same; nevertheless, we chose to distinguish them in the following, exclusively for reasons of simplicity.

Below we define the composition of configurations, as the union of disjoint sets of components and interactions:

Definition 2. *The composition of two configurations* $\gamma_i = (\mathcal{C}_i, I_i, \rho)$*, for* $i = 1,2$*, such that* $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$ *and* $I_1 \cap I_2 = \emptyset$*, is defined as* $\gamma_1 \bullet \gamma_2 \stackrel{\mathsf{def}}{=} (\mathcal{C}_1 \cup \mathcal{C}_2, I_1 \cup I_2, \rho)$*. The composition* $\gamma_1 \bullet \gamma_2$ *is undefined if* $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$ *or* $I_1 \cap I_2 \neq \emptyset$*.*

In analogy with graphs, the *degree* of a configuration is the maximum number of interactions from the configuration that involve a (possibly absent) component:

Definition 3. *The* degree *of a configuration* $\gamma = (\mathcal{C}, I, \rho)$ *is defined as* $\delta(\gamma) \stackrel{\mathsf{def}}{=} \max_{c \in \mathbb{C}} \delta_c(\gamma)$*, where* $\delta_c(\gamma) \stackrel{\mathsf{def}}{=} \|\{(c_1, p_1, \ldots, c_n, p_n) \in I \mid c = c_i,\ i \in [1,n]\}\|$*.*

For instance, the configuration of the system from Fig. 1(a) has degree two.

2.2 Configuration Logic

Let $\mathbb{V}$ and $\mathbb{A}$ be countably infinite sets of *variables* and *predicates*, respectively. For each predicate $\mathsf{A} \in \mathbb{A}$, we denote its arity by $\#\mathsf{A}$. The formulæ of the *Configuration Logic* (CL) are described inductively by the following syntax:

$$\phi := \mathsf{emp} \mid [x] \mid \langle x_1.p_1, \ldots, x_n.p_n \rangle \mid x@q \mid x = y \mid x \neq y \mid \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \mid \phi * \phi \mid \exists x \,.\, \phi$$

where $x, y, x_1, \ldots \in \mathbb{V}$, $q \in \mathcal{Q}$ and $\mathsf{A} \in \mathbb{A}$. A formula $[x]$, $\langle x_1.p_1, \ldots, x_n.p_n \rangle$, $x@q$ and $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ is called a *component*, *interaction*, *state* and *predicate* atom, respectively. These formulæ are also referred to as *atoms*. The connective $*$ is called the *separating conjunction*. We use the shorthand $[x]@q \stackrel{\mathsf{def}}{=} [x] * x@q$. For instance, the formula $[x]@q * [y]@q' * \langle x.out, y.in \rangle * \langle x.in, y.out \rangle$ describes a configuration consisting of two distinct components, denoted by the values of x and y, in states q and q', respectively, and two interactions binding the *out* port of one to the *in* port of the other component.

A formula is said to be *pure* if and only if it is a separating conjunction of state atoms, equalities and disequalities. A formula with no occurrences of predicate atoms (resp. existential quantifiers) is called *predicate-free* (resp. *quantifier-free*). A variable is *free* if it does not occur within the scope of an existential quantifier ; we note $\mathrm{fv}(\phi)$ the set of free variables of ϕ. A *sentence* is a formula with no free variables. A *substitution*

$\phi[x_1/y_1 \ldots x_n/y_n]$ replaces simultaneously every free occurrence of x_i by y_i in ϕ, for all $i \in [1,n]$. Before defining the semantics of CL formulæ, we introduce the set of inductive definitions that assigns meaning to predicates:

Definition 4. *A set of inductive definitions (SID) Δ consists of* rules $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi$*, where $x_1,\ldots,x_{\#\mathsf{A}}$ are pairwise distinct variables, called* parameters*, such that* $\mathrm{fv}(\phi) \subseteq \{x_1,\ldots,x_{\#\mathsf{A}}\}$*. The rule* $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi$ defines A *and we denote by* $\mathrm{def}_\Delta(\mathsf{A})$ *the set of rules from Δ that define* A.

Note that having distinct parameters in a rule is without loss of generality, as e.g., a rule $\mathsf{A}(x_1,x_1) \leftarrow \phi$ can be equivalently written as $\mathsf{A}(x_1,x_2) \leftarrow x_1 = x_2 * \phi$. As a convention, we shall always use the names $x_1,\ldots,x_{\#\mathsf{A}}$ for the parameters of a rule that defines A.

The semantics of CL formulæ is defined by a satisfaction relation $\gamma \models^\nu_\Delta \phi$ between configurations and formulæ. This relation is parameterized by a *store* $\nu : \mathbb{V} \rightarrow \mathbb{C}$ mapping the free variables of a formula into components from the universe (possibly absent from γ) and an SID Δ. We write $\nu[x \leftarrow c]$ for the store that maps x into c and agrees with ν on all variables other than x. The definition of the satisfaction relation is by induction on the structure of formulæ, where $\gamma = (\mathcal{C}, \mathcal{I}, \rho)$ is a configuration (Definition 1):

$$\begin{array}{lll}
\gamma \models^\nu_\Delta \mathsf{emp} & \iff & \mathcal{C} = \emptyset \text{ and } \mathcal{I} = \emptyset \\
\gamma \models^\nu_\Delta [x] & \iff & \mathcal{C} = \{\nu(x)\} \text{ and } \mathcal{I} = \emptyset \\
\gamma \models^\nu_\Delta \langle x_1.p_1,\ldots,x_n.p_n\rangle & \iff & \mathcal{C} = \emptyset \text{ and } \mathcal{I} = \{(\nu(x_1),p_1,\ldots,\nu(x_n),p_n)\} \\
\gamma \models^\nu_\Delta x@q & \iff & \gamma \models^\nu_\Delta \mathsf{emp} \text{ and } \rho(\nu(x)) = q \\
\gamma \models^\nu_\Delta x \sim y & \iff & \gamma \models^\nu_\Delta \mathsf{emp} \text{ and } \nu(x) \sim \nu(y)\text{, for all } \sim \in \{=,\neq\} \\
\gamma \models^\nu_\Delta \mathsf{A}(y_1,\ldots,y_{\#\mathsf{A}}) & \iff & \gamma \models^\nu_\Delta \phi[x_1/y_1,\ldots,x_{\#\mathsf{A}}/y_{\#\mathsf{A}}]\text{, for some rule} \\
& & \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi \text{ from } \Delta \\
\gamma \models^\nu_\Delta \phi_1 * \phi_2 & \iff & \text{exist } \gamma_1,\gamma_2\text{, such that } \gamma = \gamma_1 \bullet \gamma_2 \text{ and } \gamma_i \models^\nu_\Delta \phi_i\text{, for } i = 1,2 \\
\gamma \models^\nu_\Delta \exists x \,.\, \phi & \iff & \gamma \models^{\nu[x \leftarrow c]}_\Delta \phi\text{, for some } c \in \mathbb{C}
\end{array}$$

If ϕ is a sentence, the satisfaction relation $\gamma \models^\nu_\Delta \phi$ does not depend on the store, written $\gamma \models_\Delta \phi$, in which case we say that γ is a *model* of ϕ. If ϕ is a predicate-free formula, the satisfaction relation does not depend on the SID, written $\gamma \models^\nu \phi$. A formula ϕ is *satisfiable* if and only if the sentence $\exists x_1 \ldots \exists x_n \,.\, \phi$ has a model, where $\mathrm{fv}(\phi) = \{x_1,\ldots,x_n\}$. A formula ϕ *entails* a formula ψ, written $\phi \models_\Delta \psi$ if and only if, for any configuration γ and store ν, we have $\gamma \models^\nu_\Delta \phi$ only if $\gamma \models^\nu_\Delta \psi$.

2.3 Separation Logic

Separation Logic (SL) [39] will be used in the following to prove several technical results concerning the decidability and complexity of certain decision problems for CL. For self-containment reasons, we define SL below. The syntax of SL formulæ is described by the following grammar:

$$\phi := \mathsf{emp} \mid x_0 \mapsto (x_1,\ldots,x_{\mathfrak{K}}) \mid x = y \mid x \neq y \mid \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \mid \phi * \phi \mid \exists x \,.\, \phi$$

where $x,y,x_0,x_1,\ldots \in \mathbb{V}$, $\mathsf{A} \in \mathbb{A}$ and $\mathfrak{K} \geq 1$ is an integer constant. Formulæ of SL are interpreted over finite partial functions $\mathsf{h} : \mathbb{C} \rightharpoonup_{\mathit{fin}} \mathbb{C}^{\mathfrak{K}}$, called *heaps*[1], by a satisfaction relation $\mathsf{h} \Vdash^\nu \phi$, defined inductively as follows:

[1] We use the universe $\mathbb{C}$ here for simplicity, the definition works with any countably infinite set.

$$
\begin{array}{lll}
\mathsf{h} \Vdash^{\nu}_{\Delta} \mathsf{emp} & \iff & \mathsf{h} = \emptyset \\
\mathsf{h} \Vdash^{\nu}_{\Delta} x_0 \mapsto (x_1, \ldots, x_{\mathfrak{K}}) & \iff & \mathrm{dom}(\mathsf{h}) = \{\nu(x_0)\} \text{ and } \mathsf{h}(\nu(x_0)) = \langle \nu(x_1), \ldots, \nu(x_{\mathfrak{K}}) \rangle \\
\mathsf{h} \Vdash^{\nu} \phi_1 * \phi_2 & \iff & \text{there exist } \mathsf{h}_1, \mathsf{h}_2 \text{ such that } \mathrm{dom}(\mathsf{h}_1) \cap \mathrm{dom}(\mathsf{h}_2) = \emptyset, \\
 & & \mathsf{h} = \mathsf{h}_1 \cup \mathsf{h}_2 \text{ and } \mathsf{h}_i \Vdash^{\nu}_{\Delta} \phi_i, \text{ for both } i = 1, 2
\end{array}
$$

where $\mathrm{dom}(\mathsf{h}) \stackrel{\text{def}}{=} \{c \in \mathbb{C} \mid \mathsf{h}(c) \text{ is defined}\}$ is the domain of the heap and (dis-) equalities, predicate atoms and existential quantifiers are defined same as for CL.

2.4 Decision Problems

We define the decision problems that are the focus of the upcoming sections. As usual, a decision problem is a class of yes/no queries that differ only in their input. In our case, the input consists of an SID and one or two predicates, written between square brackets.

Definition 5. *We consider the following problems, for a SID Δ and predicates $\mathsf{A}, \mathsf{B} \in \mathbb{A}$:*

1. $\mathsf{Sat}[\Delta, \mathsf{A}]$*: is the sentence* $\exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ *satisfiable for* Δ*?*
2. $\mathsf{Bnd}[\Delta, \mathsf{A}]$*: is the set* $\{\delta(\gamma) \mid \gamma \models_{\Delta} \exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})\}$ *finite?*
3. $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$*: does* $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \models_{\Delta} \exists x_{\#\mathsf{B}+1} \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{B}(x_1, \ldots, x_{\#\mathsf{B}})$ *hold?*

The size of a formula ϕ is the total number of occurrences of symbols needed to write it down, denoted by $\mathrm{size}(\phi)$. The size of a SID Δ is $\mathrm{size}(\Delta) \stackrel{\text{def}}{=} \sum_{\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta} \mathrm{size}(\phi) + \#\mathsf{A} + 1$. Other parameters of a SID Δ are:

- $\mathrm{arity}(\Delta) \stackrel{\text{def}}{=} \max\{\#\mathsf{A} \mid \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta\}$,
- $\mathrm{width}(\Delta) \stackrel{\text{def}}{=} \max\{\mathrm{size}(\phi) \mid \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta\}$,
- $\mathrm{intersize}(\Delta) \stackrel{\text{def}}{=} \max\{n \mid \langle x_1.p_1, \ldots, x_n.p_n \rangle \text{ occurs in } \phi, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta\}$.

For a decision problem $\mathsf{P}[\Delta, \mathsf{A}, \mathsf{B}]$, we consider its (k, ℓ)-bounded versions $\mathsf{P}^{(k,\ell)}[\Delta, \mathsf{A}, \mathsf{B}]$, obtained by restricting the predicates and interaction atoms occurring Δ to $\mathrm{arity}(\Delta) \leq k$ and $\mathrm{intersize}(\Delta) \leq \ell$, respectively, where k and ℓ are either positive integers or infinity. We consider, for each $\mathsf{P}[\Delta, \mathsf{A}, \mathsf{B}]$, the subproblems $\mathsf{P}^{(k,\ell)}[\Delta, \mathsf{A}, \mathsf{B}]$ corresponding to the three cases (1) $k < \infty$ and $\ell = \infty$, (2) $k = \infty$ and $\ell < \infty$, and (3) $k = \infty$ and $\ell = \infty$. As we explain next, this is because, for the decision problems considered (Definition 5), the complexity for the case $k < \infty, \ell < \infty$ matches the one for the case $k < \infty, \ell = \infty$.

Satisfiability (1) and entailment (3) arise naturally during verification of reconfiguration programs. For instance, $\mathsf{Sat}[\Delta, \phi]$ asks whether a specification ϕ of a set configurations (e.g., a pre-, post-condition, or a loop invariant) is empty or not (e.g., an empty precondition typically denotes a vacuous verification condition), whereas $\mathsf{Entl}[\Delta, \phi, \psi]$ is used as a side condition for the Hoare rule of consequence, as in e.g., the proof from Fig. 1(c). Moreover, entailments must be proved when checking inductiveness of a user-provided loop invariant.

The $\mathsf{Bnd}[\Delta, \phi]$ problem is used to check a necessary condition for the decidability of entailments i.e., $\mathsf{Entl}[\Delta, \phi, \psi]$. If $\mathsf{Bnd}[\Delta, \phi]$ has a positive answer, we can reduce the problem $\mathsf{Entl}[\Delta, \phi, \psi]$ to an entailment problem for SL, which is always interpreted over heaps of bounded degree [18]. Otherwise, the decidability status of the entailment problem is open, for configurations of unbounded degree, such as the one described by the example below.

Example 1. The following SID describes star topologies with a central controller connected to an unbounded number of workers stations:

$$\mathit{Controller}(x) \leftarrow [x] * \mathit{Worker}(x)$$
$$\mathit{Worker}(x) \leftarrow \exists y \,.\, \langle x.\mathit{out}, y.\mathit{in}\rangle * [y] * \mathit{Worker}(x) \qquad \mathit{Worker}(x) \leftarrow \quad \mathsf{emp} \quad \blacksquare$$

3 Satisfiability

We show that the satisfiability problem (Definition 5, point 1) is decidable, using a method similar to the one pioneered by Brotherston et al. [9], for checking satisfiability of inductively defined symbolic heaps in SL. We recall that a formula π is *pure* if and only if it is a separating conjunction of equalities, disequalities and state atoms. In the following, the order of terms in (dis-)equalities is not important i.e., we consider $x = y$ (resp. $x \neq y$) and $y = x$ (resp. $y \neq x$) to be the same formula.

Definition 6. *The* closure $\mathrm{cl}(\pi)$ *of a pure formula* π *is the limit of the sequence* $\pi^0, \pi^1, \pi^2, \ldots$ *such that* $\pi^0 = \pi$ *and, for each* $i \geq 0$, π^{i+1} *is obtained by joining (with* $*$*) all of the following formulæ to* π^i*:*

- $x = z$*, where* x *and* z *are the same variable, or* $x = y$ *and* $y = z$ *both occur in* π^i*,*
- $x \neq z$*, where* $x = y$ *and* $y \neq z$ *both occur in* π^i*, or*
- $y@q$*, where* $x@q$ *and* $x = y$ *both occur in* π^i*.*

Because only finitely many such formulæ can be added, the sequence of pure formulæ from Definition 6 is bound to stabilize after polynomially many steps. A pure formula is satisfiable if and only if its closure does not contain contradictory literals i.e., $x = y$ and $x \neq y$, or $x@q$ and $x@q'$, for $q \neq q' \in \mathcal{Q}$. We write $x \approx_\pi y$ (resp. $x \not\approx_\pi y$) if and only if $x = y$ (resp. $x \neq y$) occurs in $\mathrm{cl}(\pi)$ and $\mathrm{not}(x \approx_\pi y)$ (resp. $\mathrm{not}(x \not\approx_\pi y)$) whenever $x \approx_\pi y$ (resp. $x \not\approx_\pi y$) does not hold. Note that e.g., $\mathrm{not}(x \approx_\pi y)$ is not the same as $x \not\approx_\pi y$.

Base tuples constitute the abstract domain used by the algorithms for checking satisfiability (point 1 of Definition 5) and boundedness (point 2 of Definition 5), defined as follows:

Definition 7. *A* base tuple *is a triple* $\mathfrak{t} = (\mathcal{C}^\sharp, \mathcal{I}^\sharp, \pi)$*, where:*

- $\mathcal{C}^\sharp \in \mathit{mpow}\mathbb{V}$ *is a multiset of variables denoting present components,*
- $\mathcal{I}^\sharp : \mathsf{Inter} \rightarrow \mathit{mpow}\mathbb{V}^+$ *maps each interaction type* $\tau \in \mathsf{Inter}$ *into a multiset of tuples of variables of length* $|\tau|$ *each, and*
- π *is a pure formula.*

A base tuple is called satisfiable *if and only if* π *is satisfiable and the following hold:*

1. *for all* $x, y \in \mathcal{C}^\sharp$, $\mathrm{not}(x \approx_\pi y)$*,*
2. *for all* $\tau \in \mathsf{Inter}$, $\langle x_1, \ldots, x_{|\tau|}\rangle, \langle y_1, \ldots, y_{|\tau|}\rangle \in \mathcal{I}^\sharp(\tau)$*, there exists* $i \in [1, |\tau|]$ *such that* $\mathrm{not}(x_i \approx_\pi y_i)$*,*
3. *for all* $\tau \in \mathsf{Inter}$, $\langle x_1, \ldots, x_{\#\tau}\rangle \in \mathcal{I}^\sharp(\tau)$ *and* $1 \leq i < j \leq |\tau|$*, we have* $\mathrm{not}(x_i \approx_\pi x_j)$*.*

We denote by $\mathsf{SatBase}$ *the set of satisfiable base tuples.*

Intuitively, a base tuple is an abstract representation of a configuration, where components (resp. interactions) are represented by variables (resp. tuples of variables). Note that a base tuple $(\mathcal{C}^\sharp, \mathcal{I}^\sharp, \pi)$ is unsatisfiable if $\mathcal{C}^\sharp$ $(\mathcal{I}^\sharp)$ contains the same variable (tuple of variables) twice (for the same interaction type), hence the use of multisets in the definition of base tuples. It is easy to see that checking the satisfiability of a given base tuple $(\mathcal{C}^\sharp, \mathcal{I}^\sharp, \pi)$ can be done in time $poly(\|\mathcal{C}^\sharp\| + \sum_{\tau \in \mathsf{Inter}} \|\mathcal{I}^\sharp(\tau)\| + \mathrm{size}(\pi))$.

We define a partial *composition* operation on satisfiable base tuples, as follows:

$$(\mathcal{C}_1^\sharp, \mathcal{I}_1^\sharp, \pi_1) \otimes (\mathcal{C}_2^\sharp, \mathcal{I}_2^\sharp, \pi_2) \stackrel{\mathsf{def}}{=} (\mathcal{C}_1^\sharp \cup \mathcal{C}_2^\sharp, \mathcal{I}_1^\sharp \cup \mathcal{I}_2^\sharp, \pi_1 * \pi_2)$$

where the union of multisets is lifted to functions $\mathsf{Inter} \to \mathrm{mpow}(\mathbb{V}^+)$ in the usual way. The composition operation $\otimes$ is undefined if $(\mathcal{C}_1^\sharp, \mathcal{I}_1^\sharp, \pi_1) \otimes (\mathcal{C}_2^\sharp, \mathcal{I}_2^\sharp, \pi_2)$ is not satisfiable e.g., if $\mathcal{C}_1^\sharp \cap \mathcal{C}_2^\sharp \neq \emptyset$, $\mathcal{I}_1^\sharp(\tau) \cap \mathcal{I}_2^\sharp(\tau) \neq \emptyset$, for some $\tau \in \mathsf{Inter}$, or $\pi_1 * \pi_2$ is not satisfiable.

Given a pure formula π and a set of variables X, the projection $\pi\downarrow_X$ removes from π all atoms α, such that $\mathrm{fv}(\alpha) \not\subseteq X$. The *projection* of a base tuple $(\mathcal{C}^\sharp, \mathcal{I}^\sharp, \pi)$ on a variable set X is formally defined below:

$$(\mathcal{C}^\sharp, \mathcal{I}^\sharp, \pi)\downarrow_X \stackrel{\mathsf{def}}{=} \Big(\mathcal{C}^\sharp \cap X, \lambda\tau \,.\, \{\langle x_1, \ldots, x_{|\tau|}\rangle \in \mathcal{I}^\sharp(\tau) \mid x_1, \ldots, x_{|\tau|} \in X\}, \mathrm{cl}(\mathrm{dist}(\mathcal{I}^\sharp) * \pi)\downarrow_X\Big)$$
$$\text{where } \mathrm{dist}(\mathcal{I}^\sharp) \stackrel{\mathsf{def}}{=} \mathop{\ast}\nolimits_{\tau \in \mathsf{Inter}} \mathop{\ast}\nolimits_{\langle x_1, \ldots, x_{|\tau|}\rangle \in \mathcal{I}^\sharp(\tau)} \mathop{\ast}\nolimits_{1 \leq i < j \leq |\tau|} x_i \neq x_j$$

The *substitution* operation $(\mathcal{C}^\sharp, \mathcal{I}^\sharp, \pi)[x_1/y_1, \ldots, x_n/y_n]$ replaces simultaneously each x_i with y_i in $\mathcal{C}^\sharp$, $\mathcal{I}^\sharp$ and π, respectively. We lift the composition, projection and substitution operations to sets of satisfiable base tuples, as usual.

Next, we define the base tuple corresponding to a quantifier- and predicate-free formula $\phi = \psi * \pi$, where ψ consists of component and interaction atoms and π is pure. Since, moreover, we are interested in those components and interactions that are visible through a given indexed set of parameters $X = \{x_1, \ldots, x_n\}$, for a variable y, we denote by $\{\{y\}\}_\pi^X$ the parameter x_i with the least index, such that $y \approx_\pi x_i$, or y itself, if no such parameter exists. We define the following sets of formulæ:

$$\mathsf{Base}(\phi, X) \stackrel{\mathsf{def}}{=} \begin{cases} \{(\mathcal{C}^\sharp, \mathcal{I}^\sharp, \pi)\} & \text{, if } (\mathcal{C}^\sharp, \mathcal{I}^\sharp, \pi) \text{ is satisfiable} \\ \emptyset & \text{, otherwise} \end{cases}$$
$$\text{where } \mathcal{C}^\sharp \stackrel{\mathsf{def}}{=} \{\{\{x\}\}_\pi^X \mid [x] \text{ occurs in } \psi\}$$
$$\mathcal{I}^\sharp \stackrel{\mathsf{def}}{=} \lambda\langle p_1, \ldots, p_s\rangle . \{\langle \{\{y_1\}\}_\pi^X, \ldots, \{\{y_s\}\}_\pi^X \rangle \mid \langle y_1.p_1, \ldots, y_s.p_s\rangle \text{ occurs in } \psi\}$$

We consider a tuple of variables $\overrightarrow{\mathcal{X}}$, having a variable $\mathcal{X}(\mathsf{A})$ ranging over $\mathrm{pow}(\mathsf{SatBase})$, for each predicate A that occurs in Δ. With these definitions, each rule of Δ:

$$\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathsf{B}_1(z_1^1, \ldots, z_{\#\mathsf{B}_1}^1) * \ldots * \mathsf{B}_h(z_1^h, \ldots, z_{\#\mathsf{B}_h}^h)$$

where ϕ is a quantifier- and predicate-free formula, induces the constraint:

$$\mathcal{X}(\mathsf{A}) \supseteq \Big(\mathsf{Base}(\phi, \{x_1, \ldots, x_{\#\mathsf{A}}\}) \otimes \bigotimes_{\ell=1}^{h} \mathcal{X}(\mathsf{B}_\ell)[x_1/z_1^\ell, \ldots, x_{\#\mathsf{B}_\ell}/z_{\#\mathsf{B}_\ell}^\ell]\Big)\downarrow_{x_1, \ldots, x_{\#\mathsf{A}}} \quad (1)$$

```
input: a SID $\Delta$ output: $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp$
initially $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp := \lambda\mathsf{A}\ .\ \emptyset$
for $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m\ .\ \phi \in \Delta$, with $\phi$ quantifier- and predicate-free do
    $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A}) := \mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A}) \cup \mathsf{Base}(\phi, \{x_1,\ldots,x_{\#\mathsf{A}}\})\downarrow_{x_1,\ldots,x_{\#\mathsf{A}}}$
while $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp$ still change do
    for $\mathsf{r} : \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m\ .\ \phi * \bigast_{\ell=1}^{h} \mathsf{B}_\ell(z_1^\ell,\ldots,z_{\#\mathsf{B}_\ell}^\ell) \in \Delta$ do
        if there exist $\mathsf{t}_1 \in \mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{B}_1),\ldots,\mathsf{t}_h \in \mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{B}_h)$ then
            $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A}) := \mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A}) \cup \Big(\mathsf{Base}(\phi, \{x_1,\ldots,x_{\#\mathsf{A}}\}) \otimes \bigotimes_{\ell=1}^{h} \mathsf{t}_\ell[x_1/z_1^\ell,\ldots,x_{\#\mathsf{B}_\ell}/z_{\#\mathsf{B}_\ell}^\ell]\Big)\downarrow_{x_1,\ldots,x_{\#\mathsf{A}}}$
```

Fig. 2. Algorithm for the Computation of the Least Solution

Let $\Delta^\sharp$ be the set of such constraints, corresponding to the rules in Δ and let $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp$ be the tuple of least solutions of the constraint system generated from Δ, indexed by the tuple of predicates that occur in Δ, such that $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A})$ denotes the entry of $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp$ correponding to A. Since the composition and projection are monotonic operations, such a least solution exists and is unique. Since $\mathsf{SatBase}$ is finite, the least solution can be attained in a finite number of steps, using a Kleene iteration (see Fig. 2).

We state below the main result leading to an elementary recursive algorithm for the satisfiability problem (Theorem 1). The intuition is that, if $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A})$ is not empty, then it contains only satisfiable base tuples, from which a model of $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}})$ can be built.

Lemma 1. $\mathsf{Sat}[\Delta,\mathsf{A}]$ *has a positive answer if and only if* $\mu\overrightarrow{\mathcal{X}}.\Delta^\sharp(\mathsf{A}) \neq \emptyset$*.*

If the maximal arity of the predicates occurring in Δ is bound by a constant k, no satisfiable base tuple $(\mathcal{C}^\sharp, \mathcal{I}^\sharp, \pi)$ can have a tuple $\langle y_1,\ldots,y_{|\tau|}\rangle \in \mathcal{I}^\sharp(\tau)$, for some $\tau \in \mathsf{Inter}$, such that $|\tau| > k$, since all variables $y_1,\ldots,y_{|\tau|}$ are parameters denoting distinct components (point 3 of Definition 7). Hence, the upper bound on the size of a satisfiable base tuple is constant, in both the $k < \infty, \ell < \infty$ and $k < \infty, \ell = \infty$ cases, which are, moreover indistinguishable complexity-wise (i.e., both are NP-complete). In contrast, in the cases $k = \infty, \ell < \infty$ and $k = \infty, \ell = \infty$, the upper bound on the size of satisfiable base tuples is polynomial and simply exponential in $\mathrm{size}(\Delta)$, incurring a complexity gap of one and two exponentials, respectively. The theorem below states the main result of this section:

Theorem 1. $\mathsf{Sat}^{(k,\infty)}[\Delta,\mathsf{A}]$ *is* NP*-complete for* $k \geq 4$*,* $\mathsf{Sat}^{(\infty,\ell)}[\Delta,\mathsf{A}]$ *is* EXP*-complete and* $\mathsf{Sat}[\Delta,\mathsf{A}]$ *is in* $2\mathsf{EXP}$*.*

The upper bounds are consequences of the fact that the size of a satisfiable base tuple is bounded by a simple exponential in the $\min(\mathrm{arity}(\Delta), \mathrm{intersize}(\Delta))$, hence the number of such tuples is doubly exponential in $\min(\mathrm{arity}(\Delta), \mathrm{intersize}(\Delta))$. The lower bounds are by a polynomial reduction from the satisfiability problem for SL [9].

Example 2. The doubly-exponential upper bound for the algorithm computing the least solution of a system of constraints of the form (1) is necessary, in general, as illustrated by the following worst-case example. Let n be a fixed parameter and consider the n-arity predicates $A_1,\ldots,A_n$ defined by the following SID:

$$A_i(x_1,\ldots,x_n) \leftarrow \mathop{\Huge *}\nolimits_{j=0}^{n-i} A_{i+1}(x_1,\ldots,x_{i-1},[x_i,\ldots,x_n]^j), \quad \text{for all } i \in [1,n-1]$$
$$A_n(x_1,\ldots,x_n) \leftarrow \langle x_1.p,\ldots,x_n.p\rangle \quad A_n(x_1,\ldots,x_n) \leftarrow \mathsf{emp}$$

where, for a list of variables $x_i,\ldots,x_n$ and an integer $j \geq 0$, we write $[x_i,\ldots,x_n]^j$ for the list rotated to the left j times (e.g., $[x_1,x_2,x_3,x_4,x_5]^2 = x_3,x_4,x_5,x_1,x_2$). In this example, when starting with $A_1(x_1,\ldots,x_n)$ one eventually obtains predicate atoms $A_n(x_{i_1},\ldots,x_{i_n})$, for any permutation $x_{i_1},\ldots,x_{i_n}$ of $x_1,\ldots,x_n$. Since A_n may choose to create or not an interaction with that permutation of variables, the total number of base tuples generated for A_1 is $2^{n!}$. That is, the fixpoint iteration generates $2^{2^{O(n\log n)}}$ base tuples, whereas the size of the input of $\mathsf{Sat}[\Delta,\mathsf{A}]$ is $poly(n)$. ■

4 Degree Boundedness

The boundedness problem (Definition 5, point 2) asks for the existence of a bound on the degree (Definition 3) of the models of a sentence $\exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}})$. Intuitively, the $\mathsf{Bnd}[\Delta,\mathsf{A}]$ problem has a negative answer if and only if there are increasingly large unfoldings (i.e., expansions of a formula by replacement of a predicate atom with one of its definitions) of $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}})$ repeating a rule that contains an interaction atom involving a parameter of the rule, which is always bound to the same component. We formalize the notion of unfolding below:

Definition 8. *Given a predicate* A *and a sequence* $(\mathsf{r}_1,i_1),\ldots,(\mathsf{r}_n,i_n) \in (\Delta\times\mathbb{N})^+$, *where* $\mathsf{r}_1 : \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta$, *the* unfolding $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \xRightarrow{(\mathsf{r}_1,i_1)\ldots(\mathsf{r}_n,i_n)}_{\Delta} \psi$ *is inductively defined as (1)* $\psi = \phi$ *if* $n = 1$, *and (2)* ψ *is obtained from* ϕ *by replacing its* i_1*-th predicate atom* $\mathsf{B}(y_1,\ldots,y_{\#\mathsf{B}})$ *with* $\psi_1[x_1/y_1,\ldots,x_{\#\mathsf{B}}/y_{\#\mathsf{B}}]$, *where* $\mathsf{B}(x_1,\ldots,x_{\#\mathsf{B}}) \xRightarrow{(\mathsf{r}_2,i_2)\ldots(\mathsf{r}_n,i_n)}_{\Delta} \psi_1$ *is an unfolding, if* $n > 1$.

We show that the $\mathsf{Bnd}[\Delta,\mathsf{A}]$ problem can be reduced to the existence of increasingly large unfoldings or, equivalently, a cycle in a finite directed graph, built by a variant of the least fixpoint iteration algorithm used to solve the satisfiability problem (Fig. 3).

Definition 9. *Given satisfiable base pairs* $\mathfrak{t},\mathfrak{u} \in \mathsf{SatBase}$ *and a rule from* Δ*:*

$$\mathsf{r} \;:\; \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathsf{B}_1(z^1_1,\ldots,z^1_{\#\mathsf{B}_1}) * \ldots * \mathsf{B}_h(z^h_1,\ldots,z^h_{\#\mathsf{B}_h})$$

where ϕ *is a quantifier- and predicate-free formula, we write* $(\mathsf{A},\mathfrak{t}) \overset{(\mathsf{r},i)}{\rightsquigarrow} (\mathsf{B},\mathfrak{u})$ *if and only if* $\mathsf{B} = \mathsf{B}_i$ *and there exist satisfiable base tuples* $\mathfrak{t}_1,\ldots,\mathfrak{u} = \mathfrak{t}_i,\ldots,\mathfrak{t}_h \in \mathsf{SatBase}$, *such that* $\mathfrak{t} \in \left(\mathsf{Base}(\phi,\{x_1,\ldots,x_{\#\mathsf{A}}\}) \otimes \bigotimes_{\ell=1}^{h} \mathfrak{t}_\ell[x_1/z^\ell_1,\ldots,x_{\#\mathsf{B}_\ell}/z^\ell_{\#\mathsf{B}_\ell}]\right)\downarrow_{x_1,\ldots,x_{\#\mathsf{A}}}$. *We define the directed graph with edges labeled by pairs* $(\mathsf{r},i) \in \Delta\times\mathbb{N}$*:*

$$\mathcal{G}(\Delta) \stackrel{\mathsf{def}}{=} \left(\{\mathrm{def}(\Delta)\times\mathsf{SatBase}\}, \{\langle(\mathsf{A},\mathfrak{t}),(\mathsf{r},i),(\mathsf{B},\mathfrak{u})\rangle \mid (\mathsf{A},\mathfrak{t}) \overset{(\mathsf{r},i)}{\rightsquigarrow} (\mathsf{B},\mathfrak{u})\}\right)$$

The graph $\mathcal{G}(\Delta)$ is built by the algorithm in Fig. 3, a slight variation of the classical Kleene iteration algorithm for the computation of the least solution of the constraints of the form (1). A path $(\mathsf{A}_1,\mathfrak{t}_1) \overset{(\mathsf{r}_1,i_1)}{\rightsquigarrow} (\mathsf{A}_2,\mathfrak{t}_2) \overset{(\mathsf{r}_2,i_2)}{\rightsquigarrow} \ldots \overset{(\mathsf{r}_n,i_n)}{\rightsquigarrow} (\mathsf{A}_n,\mathfrak{t}_n)$ in $\mathcal{G}(\Delta)$ induces a unique

```
input: a SID $\Delta$ output: $\mathcal{G}(\Delta) = (V, E)$
initially $V := \emptyset$, $E := \emptyset$
for $\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi \in \Delta$, with $\phi$ quantifier- and predicate-free do
    $V := V \cup \big(\{\mathsf{A}\} \times \mathsf{Base}(\phi, \{x_1,\ldots,x_{\#\mathsf{A}}\})\downarrow_{x_1,\ldots,x_{\#\mathsf{A}}}\big)$
while $V$ or $E$ still change do
    for $\mathsf{r} : \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathop{\Large\ast}_{\ell=1}^{h} \mathsf{B}_\ell(z_1^\ell,\ldots,z_{\#\mathsf{B}_\ell}^\ell) \in \Delta$ do
        if there exist $(\mathsf{B}_1,\mathfrak{t}_1),\ldots,(\mathsf{B}_h,\mathfrak{t}_h) \in V$ then
            $\mathcal{X} := \Big(\mathsf{Base}(\phi, \{x_1,\ldots,x_{\#\mathsf{A}}\}) \otimes \bigotimes_{\ell=1}^{h} \mathfrak{t}_\ell[x_1/z_1^\ell,\ldots,x_{\#\mathsf{B}_\ell}/z_{\#\mathsf{B}_\ell}^\ell]\Big)\downarrow_{x_1,\ldots,x_{\#\mathsf{A}}}$
            $V := V \cup (\{\mathsf{A}\} \times \mathcal{X})$
            $E := E \cup \{\langle(\mathsf{A},\mathfrak{t}),(\mathsf{r},\ell),(\mathsf{B}_\ell,\mathfrak{t}_\ell)\rangle \mid \mathfrak{t} \in \mathcal{X}, \ell \in [1,h]\}$
```

Fig. 3. Algorithm for the Construction of $\mathcal{G}(\Delta)$

unfolding $\mathsf{A}_1(x_1,\ldots,x_{\#\mathsf{A}_1}) \xRightarrow{(\mathsf{r}_1,i_1)\ldots(\mathsf{r}_n,i_n)}_{\Delta} \phi$ (Definition 8). Since the vertices of $\mathcal{G}(\Delta)$ are pairs $(\mathsf{A},\mathfrak{t})$, where $\mathfrak{t}$ is a satisfiable base tuple and the edges of $\mathcal{G}(\Delta)$ reflect the construction of the base tuples from the least solution of the constraints (1), the outcome ϕ of this unfolding is always a satisfiable formula.

An *elementary cycle* of $\mathcal{G}(\Delta)$ is a path from some vertex $(\mathsf{B},\mathfrak{u})$ back to itself, such that $(\mathsf{B},\mathfrak{u})$ does not occur on the path, except at its endpoints. The cycle is, moreover, *reachable* from $(\mathsf{A},\mathfrak{t})$ if and only if there exists a path $(\mathsf{A},\mathfrak{t}) \overset{(\mathsf{r}_1,i_1)}{\rightsquigarrow} \ldots \overset{(\mathsf{r}_n,i_n)}{\rightsquigarrow} (\mathsf{B},\mathfrak{u})$ in $\mathcal{G}(\Delta)$. We reduce the complement of the $\mathsf{Bnd}[\Delta,\mathsf{A}]$ problem, namely the existence of an infinite set of models of $\exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}})$ of unbounded degree, to the existence of a reachable elementary cycle in $\mathcal{G}(\Delta')$, where Δ' is obtained from Δ, as described in the following.

First, we consider, for each predicate $\mathsf{B} \in \mathrm{def}(\Delta)$, a predicate B', of arity $\#\mathsf{B}+1$, not in $\mathrm{def}(\Delta)$ i.e., the set of predicates for which there exists a rule in Δ. Second, for each rule $\mathsf{B}_0(x_1,\ldots,x_{\#\mathsf{B}_0}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathop{\Large\ast}_{\ell=2}^{h} \mathsf{B}_\ell(z_1^\ell,\ldots,z_{\#\mathsf{B}_\ell}^\ell) \in \Delta$, where ϕ is a quantifier- and predicate-free formula and $\mathrm{iv}(\phi) \subseteq \mathrm{fv}(\phi)$ denotes the subset of variables occurring in interaction atoms in ϕ, the SID Δ' has the following rules:

$$\begin{aligned}\mathsf{B}_0'(x_1,\ldots,x_{\#\mathsf{B}_0},x_{\#\mathsf{B}_0+1}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathop{\Large\ast}_{\xi \in \mathrm{iv}(\phi)} x_{\#\mathsf{B}_0+1} \neq \xi\, * \\ \mathop{\Large\ast}_{\ell=2}^{h} \mathsf{B}_\ell'(z_1^\ell,\ldots,z_{\#\mathsf{B}_\ell}^\ell,x_{\#\mathsf{B}_0+1})\end{aligned} \tag{2}$$

$$\begin{aligned}\mathsf{B}_0'(x_1,\ldots,x_{\#\mathsf{B}_0},x_{\#\mathsf{B}_0+1}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * x_{\#\mathsf{B}_0+1} = \xi\, * \\ \mathop{\Large\ast}_{\ell=2}^{h} \mathsf{B}_\ell'(z_1^\ell,\ldots,z_{\#\mathsf{B}_\ell}^\ell,x_{\#\mathsf{B}_0+1})\end{aligned} \tag{3}$$

for each variable $\xi \in \mathrm{iv}(\phi)$, that occurs in an interaction atom in ϕ.

There exists a family of models (with respect to Δ) of $\exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}})$ of unbounded degree if and only if these are models of $\exists x_1 \ldots \exists x_{\#\mathsf{A}+1} \,.\, \mathsf{A}'(x_1,\ldots,x_{\#\mathsf{A}+1})$ (with respect to Δ') and the last parameter of each predicate $\mathsf{B}' \in \mathrm{def}(\Delta')$ can be mapped, in each of the these models, to a component that occurs in unboundedly many interactions. The latter condition is equivalent to the existence of an elementary cycle, containing a rule of the form (3), that it, moreover, reachable from some vertex $(\mathsf{A}',\mathfrak{t})$ of $\mathcal{G}(\Delta')$, for some $\mathfrak{t} \in \mathsf{SatBase}$. This reduction is formalized below:

Lemma 2. *There exists an infinite sequence of configurations* $\gamma_1, \gamma_2, \ldots$ *such that* $\gamma_i \models_\Delta \exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ *and* $\delta(\gamma_i) < \delta(\gamma_{i+1})$*, for all* $i \geq 1$ *if and only if* $\mathcal{G}(\Delta')$ *has an elementary cycle containing a rule (3), reachable from a node* $(\mathsf{A}', \mathfrak{t})$*, for* $\mathfrak{t} \in \mathsf{SatBase}$*.*

The complexity result below uses a similar argument on the maximal size of (hence the number of) base tuples as in Theorem 1, leading to similar complexity gaps:

Theorem 2. $\mathsf{Bnd}^{(k,\infty)}[\Delta, \mathsf{A}]$ *is in* co-NP*,* $\mathsf{Bnd}^{(\infty,\ell)}[\Delta, \mathsf{A}]$ *is in* EXP*,* $\mathsf{Bnd}[\Delta, \mathsf{A}]$ *is in* 2EXP*.*

Moreover, the construction of $\mathcal{G}(\Delta')$ allows to prove the following cut-off result:

Proposition 1. *Let* γ *be a configuration and* ν *be a store, such that* $\gamma \models^\nu_\Delta \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$*. If* $\mathsf{Bnd}^{(k,\ell)}[\Delta, \mathsf{A}]$ *then (1)* $\delta(\gamma) = poly(\mathrm{size}(\Delta))$ *if* $k < \infty$*,* $\ell = \infty$*, (2)* $\delta(\gamma) = 2^{poly(\mathrm{size}(\Delta))}$ *if* $k = \infty$*,* $\ell < \infty$ *and (3)* $\delta(\gamma) = 2^{2^{poly(\mathrm{size}(\Delta))}}$ *if* $k = \infty$*,* $\ell = \infty$*.*

5 Entailment

This section is concerned with the entailment problem $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$, that asks whether $\gamma \models^\nu_\Delta \exists x_{\#\mathsf{A}+1} \ldots \exists x_{\#\mathsf{B}} \,.\, \mathsf{B}(x_1, \ldots, x_{\#\mathsf{B}})$, for every configuration γ and store ν, such that $\gamma \models^\nu_\Delta \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$. For instance, the proof from Fig. 1(c) relies on the following entailments, that occur as the side conditions of the Hoare logic rule of consequence:

$$\mathsf{ring}_{h,t}(y) \models_\Delta \exists x \exists z . [y]@\mathsf{H} * \langle y.out, z.in\rangle * \mathsf{chain}_{h-1,t}(z,x) * \langle x.out, y.in\rangle$$
$$[z]@\mathsf{H} * \langle z.out, x.in\rangle * \mathsf{chain}_{h-1,t}(x,y) * \langle y.out, z.in\rangle \models_\Delta \mathsf{ring}_{h,t}(z)$$

By introducing two fresh predicates A_1 and A_2, defined by the rules:

$$\mathsf{A}_1(x_1) \leftarrow \exists y \exists z . [x_1]@\mathsf{H} * \langle x_1.out, z.in\rangle * \mathsf{chain}_{h-1,t}(z,y) * \langle y.out, x_1.in\rangle \quad (4)$$
$$\mathsf{A}_2(x_1, x_2) \leftarrow \exists z . [x_1]@\mathsf{H} * \langle x_1.out, z.in\rangle * \mathsf{chain}_{h-1,t}(z,x_2) * \langle x_2.out, x_1.in\rangle \quad (5)$$

the above entailments are equivalent to $\mathsf{Entl}[\Delta, \mathsf{ring}_{h,t}, \mathsf{A}_1]$ and $\mathsf{Entl}[\Delta, \mathsf{A}_2, \mathsf{ring}_{h,t}]$, respectively, where Δ consists of the rules (4) and (5), together with the rules that define the $\mathsf{ring}_{h,t}$ and $\mathsf{chain}_{h,t}$ predicates (Sect. 1.1).

We show that the entailment problem is undecidable, in general (Thm. 3), and recover a decidable fragment, by means of three syntactic conditions, typically met in our examples. These conditions use the following notion of *profile*:

Definition 10. *The* profile *of a SID* Δ *is the pointwise greatest function* $\lambda_\Delta : \mathbb{A} \to pow(\mathbb{N})$*, mapping each predicate* A *into a subset of* $[1, \#\mathsf{A}]$*, such that, for each rule* $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi$ *from* Δ*, each atom* $\mathsf{B}(y_1, \ldots, y_{\#\mathsf{B}})$ *from* ϕ *and each* $i \in \lambda_\Delta(\mathsf{B})$*, there exists* $j \in \lambda_\Delta(\mathsf{A})$*, such that* x_j *and* y_i *are the same variable.*

The profile identifies the parameters of a predicate that are always replaced by a variable $x_1, \ldots, x_{\#\mathsf{A}}$ in each unfolding of $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$, according to the rules in Δ; it is computed by a greatest fixpoint iteration, in time $poly(\mathrm{size}(\Delta))$.

Definition 11. *A rule* $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathop{\Large *}_{\ell=1}^{h} \mathsf{B}_\ell(z_1^\ell, \ldots, z_{\#\mathsf{B}_\ell}^\ell)$*, where* ϕ *is a quantifier- and predicate-free formula, is said to be:*

1. progressing *if and only if* $\phi = [x_1] * \psi$*, where* ψ *consists of interaction atoms involving* x_1 *and (dis-)equalities, such that* $\bigcup_{\ell=1}^{h}\{z_1^{\ell},\dots,z_{\#\mathsf{B}_\ell}^{\ell}\} = \{x_2,\dots,x_{\#\mathsf{A}}\} \cup \{y_1,\dots,y_m\}$*,*
2. connected *if and only if, for each* $\ell \in [1,h]$ *there exists an interaction atom in* ψ *that contains both* z_1^{ℓ} *and a variable from* $\{x_1\} \cup \{x_i \mid i \in \lambda_\Delta(\mathsf{A})\}$*,*
3. equationally-restricted (e-restricted) *if and only if, for every disequation* $x \neq y$ *from* ϕ*, we have* $\{x,y\} \cap \{x_i \mid i \in \lambda_\Delta(\mathsf{A})\} \neq \emptyset$*.*

A SID Δ *is* progressing, connected *and* e-restricted *if and only if each rule in* Δ *is* progressing, connected *and* e-restricted*, respectively.*

For example, the SID consisting of the rules from Sect. 1.1, together with rules (4) and (5) is progressing, connected and e-restricted.

We recall that $\mathrm{def}_\Delta(\mathsf{A})$ is the set of rules from Δ that define A and denote by $\mathrm{def}^*_\Delta(\mathsf{A})$ the least superset of $\mathrm{def}_\Delta(\mathsf{A})$ containing the rules that define a predicate from a rule in $\mathrm{def}^*_\Delta(\mathsf{A})$. The following result shows that the entailment problem becomes undecidable as soon as the connectivity condition is even slightly lifted:

Theorem 3. $\mathsf{Entl}[\Delta,\mathsf{A},\mathsf{B}]$ *is undecidable, even when* Δ *is progressing and e-restricted, and only the rules in* $\mathrm{def}^*_\Delta(\mathsf{A})$ *are connected (the rules in* $\mathrm{def}^*_\Delta(\mathsf{B})$ *may be disconnected).*

On the positive side, we prove that $\mathsf{Entl}[\Delta,\mathsf{A},\mathsf{B}]$ is decidable, if Δ is progressing, connected and e-restricted, assuming further that $\mathsf{Bnd}[\Delta,\mathsf{A}]$ has a positive answer. In this case, the bound on the degree of the models of $\mathsf{A}(x_1,\dots,x_{\#\mathsf{A}})$ is effectively computable, using the algorithm from Fig. 3 (see Proposition 1 for a cut-off result) and denote by $\mathfrak{B}$ this bound, throughout this section.

The proof uses a reduction of $\mathsf{Entl}[\Delta,\mathsf{A},\mathsf{B}]$ to a similar problem for SL, showed to be decidable [18]. We recall the definition of SL, interpreted over heaps $\mathsf{h} : \mathbb{C} \rightharpoonup_{fin} \mathbb{C}^{\mathfrak{K}}$, introduced in Sect. 2.3. SL rules are denoted as $\overline{\mathsf{A}}(x_1,\dots,x_{\#(\overline{\mathsf{A}})}) \leftarrow \phi$, where ϕ is a SL formula, such that $\mathrm{fv}(\phi) \subseteq \{x_1,\dots,x_{\#(\overline{\mathsf{A}})}\}$ and SL SIDs are denoted as $\overline{\Delta}$. The profile $\lambda_{\overline{\Delta}}$ is defined for SL same as for CL (Definition 10).

Definition 12. *A* SL *rule* $\overline{\mathsf{A}}(x_1,\dots,x_{\#(\overline{\mathsf{A}})}) \leftarrow \phi$ *from a SID* $\overline{\Delta}$ *is said to be:*

1. progressing *if and only if* $\phi = \exists t_1 \dots \exists t_m \,.\, x_1 \mapsto (y_1,\dots,y_{\mathfrak{K}}) * \psi$*, where* ψ *contains only predicate and equality atoms,*
2. connected *if and only if* $z_1 \in \{x_i \mid i \in \lambda_{\overline{\Delta}}(\overline{\mathsf{A}})\} \cup \{y_1,\dots,y_{\mathfrak{K}}\}$*, for every predicate atom* $\overline{\mathsf{B}}(z_1,\dots,z_{\#(\overline{\mathsf{B}})})$ *from* ϕ*.*

Note that the definitions of progressing and connected rules are different for SL, compared to CL (Definition 11); in the rest of this section, we rely on the context to distinguish progressing (connected) SL rules from progressing (connected) CL rules. Moreover, e-restricted rules are defined in the same way for CL and SL (point 3 of Definition 11). A tight upper bound on the complexity of the entailment problem between SL formulæ, interpreted by progressing, connected and e-restricted SIDs, is given below:

Theorem 4 **([18]).** *The* SL *entailment problem is in* $2^{2^{poly(\mathrm{width}(\overline{\Delta})\cdot\log\mathrm{size}(\overline{\Delta}))}}$*, for progressing, connected and e-restricted SIDs.*

The reduction of $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$ to SL entailments is based on the idea of viewing a configuration as a logical structure (hypergraph), represented by a undirected *Gaifman graph*, in which every tuple from a relation (hyperedge) becomes a clique [24]. In a similar vein, we encode a configuration, of degree at most $\mathfrak{B}$, by a heap of degree $\mathfrak{K}$ (Definition 13), such that $\mathfrak{K}$ is defined using the following integer function:

$$\mathrm{pos}(i,j,k) \stackrel{\mathsf{def}}{=} 1 + \mathfrak{B} \cdot \sum_{\ell=1}^{j-1} |\tau_\ell| + i \cdot |\tau_j| + k$$

where $\mathsf{Inter} \stackrel{\mathsf{def}}{=} \{\tau_1, \ldots, \tau_M\}$ is the set of interaction types and $\mathcal{Q} \stackrel{\mathsf{def}}{=} \{q_1, \ldots, q_N\}$ is the set of states of the behavior $\mathbb{B} = (\mathcal{P}, \mathcal{Q}, \rightarrow)$ (Sect. 2). Here $i \in [0, \mathfrak{B} - 1]$ denotes an interaction of type $j \in [1, M]$ and $k \in [0, N-1]$ denotes a state. We use M and N throughout the rest of this section, to denote the number of interaction types and states, respectively.

For a set I of interactions, let $\mathsf{Tuples}^j_I(c) \stackrel{\mathsf{def}}{=} \{\langle c_1, \ldots, c_n \rangle \mid (c_1, p_1, \ldots, c_n, p_n) \in I,\ \tau_j = \langle p_1, \ldots, p_n \rangle,\ c \in \{c_1, \ldots, c_n\}\}$ be the tuples of components from an interaction of type τ_j from I, that contain a given component c.

Definition 13. *Given a configuration $\gamma = (\mathcal{C}, I, \rho)$, such that $\delta(\gamma) \leq \mathfrak{B}$, a* Gaifman heap *for γ is a heap $\mathsf{h} : \mathbb{C} \rightharpoonup_{fin} \mathbb{C}^{\mathfrak{K}}$, where $\mathfrak{K} \stackrel{\mathsf{def}}{=} \mathrm{pos}(0, M+1, N)$, $\mathrm{dom}(\mathsf{h}) = \mathrm{nodes}(\gamma)$ and, for all $c_0 \in \mathrm{dom}(\mathsf{h})$, such that $\mathsf{h}(c_0) = \langle c_1, \ldots, c_{\mathfrak{K}} \rangle$, the following hold:*

1. *$c_1 = c_0$ if and only if $c_0 \in \mathcal{C}$,*
2. *for all $j \in [1, M]$, $\mathsf{Tuples}^j_I(c) = \{\mathbf{c}_1, \ldots, \mathbf{c}_s\}$ if and only if there exist integers $0 \leq k_1 < \ldots < k_s < \mathfrak{B}$, such that $\langle \mathsf{h}(c_0) \rangle_{\mathrm{inter}(k_i, j)} = \mathbf{c}_i$, for all $i \in [1, s]$, where $\mathrm{inter}(i, j) \stackrel{\mathsf{def}}{=} [\mathrm{pos}(i-1, j, 0), \mathrm{pos}(i, j, 0)]$ are the entries of the i-th interaction of type τ_j in $\mathsf{h}(c_0)$,*
3. *for all $k \in [1, N]$, we have $\langle \mathsf{h}(c_0) \rangle_{\mathrm{state}(k)} = c_0$ if and only if $\rho(c_0) = q_k$, where the entry $\mathrm{state}(k) \stackrel{\mathsf{def}}{=} \mathrm{pos}(0, M+1, k-1)$ in $\mathsf{h}(c_0)$ corresponds to the state $q_k \in \mathcal{Q}$.*

We denote by $\mathbb{G}(\gamma)$ the set of Gaifman heaps for γ.

Intuitively, if h is a Gaifman heap for γ and $c_0 \in \mathrm{dom}(\mathsf{h})$, then the first entry of $\mathsf{h}(c_0)$ indicates whether c_0 is present (condition 1 of Definition 13), the next $\mathfrak{B} \cdot \sum_{j=1}^{M} |\tau_j|$ entries are used to encode the interactions of each type τ_j (condition 2 of Definition 13), whereas the last N entries are used to represent the state of the component (condition 3 of Definition 13). Note that the encoding of configurations by Gaifman heaps is not unique: two Gaifman heaps for the same configuration may differ in the order of the tuples from the encoding of an interaction type and the choice of the unconstrained entries from $\mathsf{h}(c_0)$, for each $c_0 \in \mathrm{dom}(\mathsf{h})$. On the other hand, if two configurations have the same Gaifman heap encoding, they must be the same configuration.

Example 3. Figure 4(b) shows a Gaifman heap for the configuration in Fig. 4(a), where each component belongs to at most 2 interactions of type $\langle out, in \rangle$. ■

We build a SL SID $\overline{\Delta}$ that generates the Gaifman heaps of the models of the predicate atoms occurring in a progressing CL SID Δ. The construction associates to each variable x, that occurs free or bound in a rule from Δ, a unique $\mathfrak{K}$-tuple of variables $\eta(x) \in \mathbb{V}^{\mathfrak{K}}$,

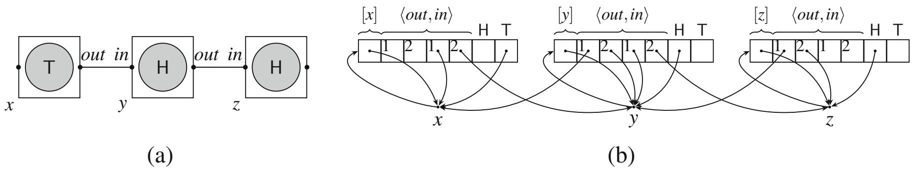


Fig. 4. Gaifman Heap for a Chain Configuration

that represents the image of the store value $\nu(x)$ in a Gaifman heap h i.e., $\mathsf{h}(\nu(x)) = \nu(\eta(x))$. Moreover, we consider, for each predicate symbol $\mathsf{A} \in \mathrm{def}(\Delta)$, an annotated predicate symbol $\overline{\mathsf{A}}_\iota$ of arity $\#\overline{\mathsf{A}}_\iota = (\mathfrak{K}+1)\cdot\#\mathsf{A}$, where $\iota : [1,\#\mathsf{A}] \times [1,M] \to 2^{[0,\mathfrak{B}-1]}$ is a map associating each parameter $i \in [1,\#\mathsf{A}]$ and each interaction type τ_j, for $j \in [1,M]$, a set of integers $\iota(i,j)$ denoting the positions of the encodings of the interactions of type τ_j, involving the value of x_i, in the models of $\overline{\mathsf{A}}_\iota(x_1,\ldots,x_{\#\mathsf{A}},\eta(x_1),\ldots,\eta(x_{\#\mathsf{A}}))$ (point 2 of Definition 13). Then $\overline{\Delta}$ contains rules of the form:

$$\overline{\mathsf{A}}_\iota(x_1,\ldots,x_{\#(\mathsf{A})},\eta(x_1),\ldots,\eta(x_{\#(\mathsf{A})})) \leftarrow \tag{6}$$
$$\exists y_1\ldots\exists y_m\exists\eta(y_1)\ldots\exists\eta(y_m)\ .\ \overline{\psi}*\pi\ *\mathop{\Large\ast}_{\ell=1}^{h}\overline{\mathsf{B}}^{\ell}_{\iota^\ell}(z^\ell_1,\ldots,z^\ell_{\#(\mathsf{B}^\ell)},\eta(z^\ell_1),\ldots,\eta(z^\ell_{\#(\mathsf{B}^\ell)}))$$

for which Δ has a *stem rule* $\mathsf{A}(x_1,\ldots,x_{\#(\mathsf{A})}) \leftarrow \exists y_1\ldots\exists y_m\ .\ \psi*\pi*\mathop{\Large\ast}_{\ell=1}^{h}\mathsf{B}^\ell(z^\ell_1,\ldots,z^\ell_{\#\mathsf{B}^\ell})$, where $\psi*\pi$ is a quantifier- and predicate-free formula and π is the conjunction of equalities and disequalities from $\psi*\pi$. However, not all rules (6) are considered in $\overline{\Delta}$, but only the ones meeting the following condition:

Definition 14. *A rule of the form (6) is* well-formed *if and only if, for each $i \in [1,\#\mathsf{A}]$ and each $j \in [1,M]$, there exists a set of integers $Y_{i,j} \subseteq [0,\mathfrak{B}-1]$, such that:*

- *$\|Y_{i,j}\| = \|I^j_{\psi,\pi}(x_i)\|$, where $I^j_{\psi,\pi}(x)$ is the set of interaction atoms $\langle z_1.p_1,\ldots,z_n.p_n\rangle$ from ψ of type $\tau_j = \langle p_1,\ldots,p_n\rangle$, such that $z_s \approx_\pi x$, for some $s \in [1,n]$,*
- *$Y_{i,j} \subseteq \iota(i,j)$ and $\iota(i,j)\backslash Y_{i,j} = \mathcal{Z}_j(x_i)$, where $\mathcal{Z}_j(x) \stackrel{\mathrm{def}}{=} \bigcup_{\ell=1}^{h}\bigcup_{k=1}^{\#\mathsf{B}^\ell}\{\iota^\ell(k,j) \mid x \approx_\pi z^\ell_k\}$ is the set of positions used to encode the interactions of type τ_j involving the store value of the parameter x, in the sub-configuration corresponding to an atom $\mathsf{B}_\ell(z^\ell_1,\ldots,z^\ell_{\#(\mathsf{B}^\ell)})$, for some $\ell \in [1,h]$.*

We denote by $\overline{\Delta}$ the set of well-formed rules (6), such that, moreover:

$$\overline{\psi} \stackrel{\mathrm{def}}{=} x_1 \mapsto \eta(x_1)\ *\ \mathop{\Large\ast}_{x\in\mathrm{fv}(\psi)}\mathsf{CompStates}_\psi(x)\ *\ \mathop{\Large\ast}_{i=1}^{\#\mathsf{A}}\mathsf{InterAtoms}_\psi(x_i)\text{, where:}$$
$$\mathsf{CompStates}_\psi(x) \stackrel{\mathrm{def}}{=} \mathop{\Large\ast}_{[x]\text{ occurs in }\psi}\langle\eta(x)\rangle_1 = x\ *\ \mathop{\Large\ast}_{x@q_k\text{ occurs in }\psi}\langle\eta(x)\rangle_{\mathrm{state}(k)} = x$$
$$\mathsf{InterAtoms}_\psi(x_i) \stackrel{\mathrm{def}}{=} \mathop{\Large\ast}_{j=1}^{M}\mathop{\Large\ast}_{p=1}^{r_j}\langle\eta(x_i)\rangle_{\mathrm{inter}(j,k^j_p)} = \mathbf{x}^j_p \text{ and } \{k^j_1,\ldots,k^j_{r_j}\} \stackrel{\mathrm{def}}{=} \iota(i,j)\backslash\mathcal{Z}_j(x_i)$$

Here for two tuples of variables $\mathbf{x} = \langle x_1,\ldots,x_k\rangle$ and $\mathbf{y} = \langle y_1,\ldots,y_k\rangle$, we denote by $\mathbf{x} = \mathbf{y}$ the formula $\mathop{\Large\ast}_{i=1}^{k} x_i = y_i$. Intuitively, the SL formula $\mathsf{CompStates}_\psi(x)$ realizes the encoding of the component and state atoms from ψ, in the sense of points (1) and (3) from Definition 13, whereas the formula $\mathsf{InterAtoms}_\psi(x_i)$ realizes the encodings of the interactions involving a parameter x_i in the stem rule (point 2 of Definition 13). In particular, the definition of $\mathsf{InterAtoms}_\psi(x_i)$ uses the fact that the rule is well-formed.

We state below the main result of this section on the complexity of the entailment problem. The upper bounds follow from a many-one reduction of $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$ to the SL entailment $\overline{\mathsf{A}}_\iota(x_1, \ldots, x_{\#\mathsf{A}}, \eta(x_1), \ldots, \eta(x_{\#\mathsf{A}})) \Vdash_{\overline{\Delta}} \exists x_{\#\mathsf{B}+1} \ldots \exists x_{\#\mathsf{B}}$ $\exists \eta(x_{\#\mathsf{B}+1}) \ldots \exists \eta(x_{\#\mathsf{B}})$. $\overline{\mathsf{B}}_{\iota'}(x_1, \ldots, x_{\#\mathsf{B}}, \eta(x_1), \ldots, \eta(x_{\#\mathsf{B}}))$, in combination with the upper bound provided by Theorem 4, for SL entailments. If $k < \infty$, the complexity is tight for CL, whereas gaps occur for $k = \infty, \ell < \infty$ and $k = \infty, \ell = \infty$, due to the cut-off on the degree bound (Proposition 1), which impacts the size of $\overline{\Delta}$ and time needed to generate it from Δ.

Theorem 5. *If Δ is progressing, connected and e-restricted and, moreover,* $\mathsf{Bnd}[\Delta, \mathsf{A}]$ *has a positive answer,* $\mathsf{Entl}^{k,\ell}[\Delta, \mathsf{A}, \mathsf{B}]$ *is in* 2EXP, $\mathsf{Entl}^{\infty,\ell}[\Delta, \mathsf{A}, \mathsf{B}]$ *is in* 3EXP ∩ 2EXP-*hard, and* $\mathsf{Entl}[\Delta, \mathsf{A}, \mathsf{B}]$ *is in* 4EXP ∩ 2EXP-*hard.*

6 Conclusions and Future Work

We study the satisfiability and entailment problems in a logic used to write proofs of correctness for dynamically reconfigurable distributed systems. The logic views the components and interactions from the network as resources and reasons also about the local states of the components. We reuse existing techniques for Separation Logic [39], showing that our configuration logic is more expressive than SL, fact which is confirmed by a number of complexity gaps. Closing up these gaps and finding tight complexity classes in the more general cases is considered for future work. In particular, we aim at lifting the boundedness assumption on the degree of the configurations that must be considered to check the validity of entailments.

References

1. Ahrens, E., Bozga, M., Iosif, R., Katoen, J.: Local reasoning about parameterized reconfigurable distributed systems. CoRR, abs/2107.05253 (2021)
2. Arbab, F.: Reo: a channel-based coordination model for component composition. Math. Struct. Comput. Sci. **14**(3), 329–366 (2004)
3. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006), pp. 3–12. IEEE Computer Society (2006)
4. Bloem, R., et al.: Decidability of Parameterized Verification. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers (2015)
5. Bozga, M., Bueri, L., Iosif, R.: Decision problems in a logic for reasoning about reconfigurable distributed systems. CoRR, abs/2202.09637 (2022)
6. Bozga, M., Iosif, R., Sifakis, J.: Verification of component-based systems with recursive architectures. CoRR, abs/2112.08292 (2021)
7. Bradbury, J., Cordy, J., Dingel, J., Wermelinger, M.: A survey of self-management in dynamic software architecture specifications. In: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems, pp. 28–33. ACM (2004)
8. Brookes, S., O'Hearn, P.W.: Concurrent separation logic. ACM SIGLOG News **3**(3), 47–65 (2016)

9. Brotherston, J., Fuhs, C., Pérez, J.A.N., Gorogiannis, N.: A decision procedure for satisfiability in separation logic with inductive predicates. In: CSL-LICS, pp. 25:1–25:10. ACM (2014)
10. Bucchiarone, A., Galeotti, J.P.: Dynamic software architectures verification using dynalloy. Electron. Commun. Eur. Assoc. Softw. Sci. Technol. **10** (2008). https://doi.org/10.14279/tuj.eceasst.10.145
11. Butting, A., Heim, R., Kautz, O., Ringert, J.O., Rumpe, B., Wortmann, A.: A classification of dynamic reconfiguration in component and connector architecture description. In: Proceedings of MODELS 2017 Satellite Event: Workshops (ModComp). CEUR Workshop Proceedings, vol. 2019, pp. 10–16. CEUR-WS.org (2017)
12. Calcagno, C., O'Hearn, P.W., Yang, H.: Local action and abstract separation logic. In: 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10–12 July 2007, Wroclaw, Poland, Proceedings, pp. 366–378. IEEE Computer Society (2007)
13. Cavalcante, E., Batista, T.V., Oquendo, F.: Supporting dynamic software architectures: from architectural description to implementation. In: Bass, L., Lago, P., Kruchten, P. (eds.) 12th Working IEEE/IFIP Conference on Software Architecture, WICSA 2015, pp. 31–40. IEEE Computer Society (2015)
14. Clarke, D.: A basic logic for reasoning about connector reconfiguration. Fundam. Inf. **82**(4), 361–390 (2008)
15. Dinsdale-Young, T., Birkedal, L., Gardner, P., Parkinson, M., Yang, H.: Views: compositional reasoning for concurrent programs. SIGPLAN Not. **48**(1), 287–300 (2013)
16. Dinsdale-Young, T., Dodds, M., Gardner, P., Parkinson, M.J., Vafeiadis, V.: Concurrent abstract predicates. In: D'Hondt, T. (ed.) ECOOP 2010. LNCS, vol. 6183, pp. 504–528. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14107-2_24
17. Dormoy, J., Kouchnarenko, O., Lanoix, A.: Using temporal logic for dynamic reconfigurations of components. In: Barbosa, L.S., Lumpe, M. (eds.) FACS 2010. LNCS, vol. 6921, pp. 200–217. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27269-1_12
18. Echenim, M., Iosif, R., Peltier, N.: Unifying decidable entailments in separation logic with inductive definitions. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 183–199. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79876-5_11
19. El-Ballouli, R., Bensalem, S., Bozga, M., Sifakis, J.: Programming dynamic reconfigurable systems. Int. J. Softw. Tools Technol. Transf. **23**, 701–719 (2021)
20. El-Hokayem, A., Bozga, M., Sifakis, J.: A temporal configuration logic for dynamic reconfigurable systems. In: Hung, C., Hong, J., Bechini, A., Song, E. (eds.) SAC 2021: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, 22–26 March 2021, pp. 1419–1428. ACM (2021)
21. Farka, F., Nanevski, A., Banerjee, A., Delbianco, G.A., Fábregas, I.: On algebraic abstractions for concurrent separation logics. Proc. ACM Program. Lang. **5**(POPL), 1–32 (2021)
22. Feng, X., Ferreira, R., Shao, Z.: On the relationship between concurrent separation logic and assume-guarantee reasoning. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 173–188. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71316-6_13
23. Foerster, K., Schmid, S.: Survey of reconfigurable data center networks: enablers, algorithms, complexity. SIGACT News **50**(2), 62–79 (2019)
24. Gaifman, H.: On local and non-local properties. Stud. Log. Found. Math. **107**, 105–135 (1982)
25. Gunawi, H.S., et al.: Why does the cloud stop computing? Lessons from hundreds of service outages. In: Proceedings of the Seventh ACM Symposium on Cloud Computing, SoCC 2016, pp. 1–16. Association for Computing Machinery, New York (2016)
26. Hirsch, D., Inverardi, P., Montanari, U.: Graph grammars and constraint solving for software architecture styles. In: Proceedings of the Third International Workshop on Software Architecture, ISAW 1998, pp. 69–72. Association for Computing Machinery, New York (1998)

27. Jansen, C., Katelaan, J., Matheja, C., Noll, T., Zuleger, F.: Unified reasoning about robustness properties of symbolic-heap separation logic. In: Yang, H. (ed.) ESOP 2017. LNCS, vol. 10201, pp. 611–638. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54434-1_23
28. Jones, C.B.: Developing methods for computer programs including a notion of interference. Ph.D. thesis, University of Oxford, UK (1981)
29. Konnov, I.V., Kotek, T., Wang, Q., Veith, H., Bliudze, S., Sifakis, J.: Parameterized systems in BIP: design and model checking. In: 27th International Conference on Concurrency Theory, CONCUR 2016, volume 59 of LIPIcs, pp. 30:1–30:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016)
30. Krause, C., Maraikar, Z., Lazovik, A., Arbab, F.: Modeling dynamic reconfigurations in Reo using high-level replacement systems. Sci. Comput. Program. **76**, 23–36 (2011)
31. Lanoix, A., Dormoy, J., Kouchnarenko, O.: Combining proof and model-checking to validate reconfigurable architectures. Electron. Notes Theor. Comput. Sci. **279**(2), 43–57 (2011)
32. Le Metayer, D.: Describing software architecture styles using graph grammars. IEEE Trans. Softw. Eng. **24**(7), 521–533 (1998)
33. Magee, J., Kramer, J.: Dynamic structure in software architectures. In: ACM SIGSOFT Software Engineering Notes, vol. 21, no. 6, pp. 3–14. ACM (1996)
34. Mavridou, A., Baranov, E., Bliudze, S., Sifakis, J.: Configuration logics: modeling architecture styles. J. Log. Algebr. Meth. Program. **86**(1), 2–29 (2017)
35. Noormohammadpour, M., Raghavendra, C.S.: Datacenter traffic control: understanding techniques and tradeoffs. IEEE Commun. Surv. Tutor. **20**(2), 1492–1525 (2018)
36. O'Hearn, P.W.: Resources, concurrency, and local reasoning. Theor. Comput. Sci. **375**(1–3), 271–307 (2007)
37. O'Hearn, P.W., Pym, D.J.: The logic of bunched implications. Bull. Symb. Log. **5**(2), 215–244 (1999)
38. Owicki, S., Gries, D.: An axiomatic proof technique for parallel programs. In: Gries, D. (ed.) Programming Methodology. Texts and Monographs in Computer Science, pp. 130–152. Springer, New York (1978). https://doi.org/10.1007/978-1-4612-6315-9_12
39. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: Proceedings of 17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22–25 July 2002, Copenhagen, Denmark, pp. 55–74. IEEE Computer Society (2002)
40. Shtadler, Z., Grumberg, O.: Network grammars, communication behaviors and automatic verification. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 151–165. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52148-8_13
41. Taentzer, G., Goedicke, M., Meyer, T.: Dynamic change management by distributed graph transformation: towards configurable distributed systems. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) TAGT 1998. LNCS, vol. 1764, pp. 179–193. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-540-46464-8_13
42. Vafeiadis, V., Parkinson, M.: A marriage of rely/guarantee and separation logic. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 256–271. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74407-8_18
43. Wermelinger, M.: Towards a chemical model for software architecture reconfiguration. IEE Proc.-Softw. **145**(5), 130–136 (1998)
44. Wermelinger, M., Fiadeiro, J.L.: A graph transformation approach to software architecture reconfiguration. Sci. Comput. Program. **44**(2), 133–155 (2002)

Proving Non-Termination and Lower Runtime Bounds with LoAT (System Description)

Florian Frohn(✉) and Jürgen Giesl(✉)

LuFG Informatik 2, RWTH Aachen University, Aachen, Germany
florian.frohn@cs.rwth-aachen.de, giesl@informatik.rwth-aachen.de

Abstract. We present the *Loop Acceleration Tool* (LoAT), a powerful tool for proving non-termination and worst-case lower bounds for programs operating on integers. It is based on the novel calculus from [10,11] for *loop acceleration*, i.e., transforming loops into non-deterministic straight-line code, and for finding non-terminating configurations. To implement it efficiently, LoAT uses a new approach based on unsat cores. We evaluate LoAT's power and performance by extensive experiments.

1 Introduction

Efficiency is one of the most important properties of software. Consequently, *automated complexity analysis* is of high interest to the software verification community. Most research in this area has focused on deducing *upper* bounds on the worst-case complexity of programs. In contrast, the *Loop Acceleration Tool* LoAT aims to find performance bugs by deducing *lower* bounds on the worst-case complexity of programs operating on integers. Since non-termination implies the lower bound ∞, LoAT is also equipped with non-termination techniques.

LoAT is based on *loop acceleration* [4,5,9–11,15], which replaces loops by non-deterministic code: The resulting program chooses a value n, representing the number of loop iterations in the original program. To be sound, suitable constraints on n are synthesized to ensure that the original loop allows for at least n iterations. Moreover, the transformed program updates the program variables to the same values as n iterations of the original loop, but it does so in a single step. To achieve that, the loop body is transformed into a *closed form*, which is parameterized in n. In this way, LoAT is able to compute *symbolic under-approximations* of programs, i.e., every execution path in the resulting transformed program corresponds to a path in the original program, but not necessarily vice versa. In contrast to many other techniques for computing under-approximations, the symbolic approximations of LoAT cover *infinitely many runs* of *arbitrary length*.

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 235950644 (Project GI 274/6-2).

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 712–722, 2022.
https://doi.org/10.1007/978-3-031-10769-6_41

Contributions: The main new feature of the novel version of LoAT presented in this paper is the *integration* of the *loop acceleration calculus* from [10,11], which combines different loop acceleration techniques in a modular way, into LoAT's framework. This enables LoAT to use the loop acceleration calculus for the analysis of full integer programs, whereas the standalone implementation of the calculus from [10,11] was only applicable to single loops without branching in the body. To control the application of the calculus, we use a new technique based on unsat cores (see Sect. 5). The new version of LoAT is evaluated in extensive experiments. See [14] for all proofs.

2 Preliminaries

Let $\mathcal{L} \supseteq \{main\}$ be a finite set of *locations*, where *main* is the *canonical start location* (i.e., the entry point of the program), and let $\vec{x} := [x_1, \ldots, x_d]$ be the vector of *program variables*. Furthermore, let $\mathcal{TV}$ be a countably infinite set of *temporary variables*, which are used to model non-determinism, and let $\sup \mathbb{Z} := \infty$. We call an arithmetic expression e an *integer expression* if it evaluates to an integer when all variables in e are instantiated by integers. LoAT analyzes tail-recursive programs operating on integers, represented as *integer transition systems* (ITSs), i.e., sets of *transitions* $f(\vec{x}) \xrightarrow{p} g(\vec{a})\,[\varphi]$ where $f, g \in \mathcal{L}$, the *update* $\vec{a}$ is a vector of d integer expressions over $\mathcal{TV} \cup \vec{x}$, the *cost* p is either an arithmetic expression over $\mathcal{TV} \cup \vec{x}$ or ∞, and the *guard* φ is a conjunction of inequations over integer expressions with variables from $\mathcal{TV} \cup \vec{x}$.[1] For example, consider the loop on the left and the corresponding transition t_{loop} on the right.

$$\textbf{while } x > 0 \textbf{ do } x \leftarrow x - 1 \qquad f(x) \xrightarrow{1} f(x-1)\,[x > 0] \qquad (t_{loop})$$

Here, the cost 1 instructs LoAT to use the number of loop iterations as cost measure. LoAT allows for arbitrary *user defined* cost measures, since the user can choose any polynomials over the program variables as costs. LoAT synthesizes transitions with cost ∞ to represent non-terminating runs, i.e., such transitions are not allowed in the input.

A *configuration* is of the form $f(\vec{c})$ with $f \in \mathcal{L}$ and $\vec{c} \in \mathbb{Z}^d$. For any entity $s \notin \mathcal{L}$ and any arithmetic expressions $\vec{b} = [b_1, \ldots, b_d]$, let $s(\vec{b})$ denote the result of replacing each variable x_i in s by b_i, for all $1 \leq i \leq d$. Moreover, $\mathcal{V}ars(s)$ denotes the program variables and $\mathcal{TV}(s)$ denotes the temporary variables occurring in s. For an integer transition system $\mathcal{T}$, a configuration $f(\vec{c})$ *evaluates to* $g(\vec{c}\,')$ *with cost* $k \in \mathbb{Z} \cup \{\infty\}$, written $f(\vec{c}) \xrightarrow{k}_{\mathcal{T}} g(\vec{c}\,')$, if there exist a transition $f(\vec{x}) \xrightarrow{p} g(\vec{a})\,[\varphi] \in \mathcal{T}$ and an instantiation of its temporary variables with integers such that the following holds:

$$\varphi(\vec{c}) \land \vec{c}\,' = \vec{a}(\vec{c}) \land k = p(\vec{c}).$$

[1] LoAT can also analyze the complexity of certain non-tail-recursive programs, see [9]. For simplicity, we restrict ourselves to tail-recursive programs in the current paper.

As usual, we write $f(\vec{c}) \xrightarrow{k}^{*}_{\mathcal{T}} g(\vec{c}\,')$ if $f(\vec{c})$ evaluates to $g(\vec{c}\,')$ in arbitrarily many steps, and the sum of the costs of all steps is k. We omit the costs if they are irrelevant. The *derivation height* of $f(\vec{c})$ is

$$dh_{\mathcal{T}}(f(\vec{c})) := \sup\{k \mid \exists g(\vec{c}\,').\, f(\vec{c}) \xrightarrow{k}^{*}_{\mathcal{T}} g(\vec{c}\,')\}$$

and the *runtime complexity* of $\mathcal{T}$ is

$$rc_{\mathcal{T}}(n) := \sup\{\, dh_{\mathcal{T}}(main(c_1, \ldots, c_d)) \mid |c_1| + \ldots + |c_d| \leq n\}.$$

$\mathcal{T}$ terminates if no configuration $main(\vec{c})$ admits an infinite $\rightarrow_{\mathcal{T}}$-sequence and $\mathcal{T}$ is *finitary* if no configuration $main(\vec{c})$ admits a $\rightarrow_{\mathcal{T}}$-sequence with cost ∞. Otherwise, $\vec{c}$ is a *witness of non-termination* or a *witness of infinitism*, respectively. Note that termination implies finitism for ITSs where no transition has cost ∞. However, our approach may transform non-terminating ITSs into terminating, infinitary ITSs, as it replaces non-terminating loops by transitions with cost ∞.

3 Overview of LoAT

The goal of LoAT is to compute a lower bound on $rc_{\mathcal{T}}$ or even prove non-termination of $\mathcal{T}$. To this end, it repeatedly applies program simplifications, so-called *processors*. When applying them with a suitable strategy (see [8,9]), one eventually obtains *simplified transitions* of the form $main(\vec{x}) \xrightarrow{p} f(\vec{a})\,[\varphi]$ where $f \neq main$. As LoAT's processors are *sound for lower bounds* (i.e., if they transform $\mathcal{T}$ to $\mathcal{T}'$, then $dh_{\mathcal{T}} \geq dh_{\mathcal{T}'}$), such a simplified transition gives rise to the lower bound $I_{\varphi} \cdot p$ on $dh_{\mathcal{T}}(main(\vec{x}))$ (where I_{φ} denotes the indicator function of φ, which is 1 for values where φ holds and 0 otherwise). This bound can be lifted to $rc_{\mathcal{T}}$ by solving a so-called *limit problem*, see [9].

LoAT's processors are also *sound for non-termination*, as they preserve finitism. So if $p = \infty$, then it suffices to prove satisfiability of φ to prove infinitism, which implies non-termination of the original ITS, where transitions with cost ∞ are forbidden (see Sect. 2). LoAT's most important processors are:

Loop Acceleration (Sect. 4) transforms a *simple loop*, i.e., a single transition $f(\vec{x}) \xrightarrow{p} f(\vec{a})\,[\varphi]$, into a non-deterministic transition that can simulate several loop iterations in one step. For example, loop acceleration transforms t_{loop} to

$$f(x) \xrightarrow{n} f(x-n)\,[x \geq n \wedge n > 0]\,, \qquad (t_{loop^n})$$

where $n \in \mathcal{TV}$, i.e., the value of n can be chosen non-deterministically.

Instantiation [9, Theorem 3.12] replaces temporary variables by integer expressions. For example, it could instantiate n with x in t_{loop^n}, resulting in

$$f(x) \xrightarrow{x} f(0)\,[x > 0]\,. \qquad (t_{loop^x})$$

Chaining [9, Theorem 3.18] combines two subsequent transitions into one transition. For example, chaining combines the transitions

$$\begin{aligned} & \mathit{main}(x) \xrightarrow{1} f(x) \\ \text{and } t_{loop^x} \text{ to } \quad & \mathit{main}(x) \xrightarrow{x+1} f(0)\,[x > 0]\,. \end{aligned}$$

Nonterm (Sect. 6) searches for witnesses of non-termination, characterized by a formula ψ. So it turns, e.g.,

$$f(x_1, x_2) \xrightarrow{1} f(x_1 - x_2, x_2)\,[x_1 > 0] \qquad (t_{nonterm})$$

$$\text{into} \quad f(x_1, x_2) \xrightarrow{\infty} \mathit{sink}(x_1, x_2)\,[x_1 > 0 \land x_2 \leq 0]$$

(where $\mathit{sink} \in \mathcal{L}$ is fresh), as each $\vec{c} \in \mathbb{Z}^2$ with $c_1 > 0 \land c_2 \leq 0$ witnesses non-termination of $t_{nonterm}$, i.e., here ψ is $x_1 > 0 \land x_2 \leq 0$.

Intuitively, LoAT uses **Chaining** to transform non-simple loops into simple loops. **Instantiation** resolves non-determinism heuristically and thus reduces the number of temporary variables, which is crucial for scalability. In addition to these processors, LoAT removes transitions after processing them, as explained in [9]. See [8,9] for heuristics and a suitable strategy to apply LoAT's processors.

4 Modular Loop Acceleration

For **Loop Acceleration**, LoAT uses *conditional acceleration techniques* [10]. Given two formulas ξ and $\check{\varphi}$, and a loop with update $\vec{a}$, a conditional acceleration technique yields a formula $\mathit{accel}(\xi, \check{\varphi}, \vec{a})$ which implies that ξ holds throughout n loop iterations (i.e., ξ is an *n-invariant*), provided that $\check{\varphi}$ is an n-invariant, too. In the following, let $\vec{a}^0(\vec{x}) := \vec{x}$ and $\vec{a}^{m+1}(\vec{x}) := \vec{a}(\vec{a}^m(\vec{x})) = \vec{a}[\vec{x}/\vec{a}^m(\vec{x})]$.

Definition 1 (Conditional Acceleration Technique). *A function accel is a* conditional acceleration technique *if the following implication holds for all formulas ξ and $\check{\varphi}$ with variables from $\mathcal{TV} \cup \vec{x}$, all updates $\vec{a}$, all $n > 0$, and all instantiations of the variables with integers:*

$$\big(\mathit{accel}(\xi, \check{\varphi}, \vec{a}) \land \forall i \in [0, n).\ \check{\varphi}(\vec{a}^i(\vec{x}))\big) \implies \forall i \in [0, n).\ \xi(\vec{a}^i(\vec{x})).$$

The prerequisite $\forall i \in [0, n).\ \check{\varphi}(\vec{a}^i(\vec{x}))$ is ensured by previous acceleration steps, i.e., $\check{\varphi}$ is initially $\top$ (*true*), and it is refined by conjoining a part ξ of the loop guard in each acceleration step. When formalizing acceleration techniques, we only specify the result of *accel* for certain arguments ξ, $\check{\varphi}$, and $\vec{a}$, and assume $\mathit{accel}(\xi, \check{\varphi}, \vec{a}) = \bot$ (*false*) otherwise.

Definition 2 (LoAT's Conditional Acceleration Techniques [10,11]).

$$
\begin{array}{lll}
\textbf{Increase} & \mathit{accel}_{\mathit{inc}}(\xi, \breve{\varphi}, \vec{a}) := \xi & \mathit{if} \models \xi \wedge \breve{\varphi} \implies \xi(\vec{a}) \\
\textbf{Decrease} & \mathit{accel}_{\mathit{dec}}(\xi, \breve{\varphi}, \vec{a}) := \xi(\vec{a}^{n-1}(\vec{x})) & \mathit{if} \models \xi(\vec{a}) \wedge \breve{\varphi} \implies \xi \\
\textbf{Eventual Decrease} & \multicolumn{2}{l}{\mathit{accel}_{\mathit{ev\text{-}dec}}(t > 0, \breve{\varphi}, \vec{a}) := t > 0 \wedge t(\vec{a}^{n-1}(\vec{x})) > 0} \\
 & & \mathit{if} \models (t \geq t(\vec{a}) \wedge \breve{\varphi}) \implies t(\vec{a}) \geq t(\vec{a}^2(\vec{x})) \\
\textbf{Eventual Increase} & \multicolumn{2}{l}{\mathit{accel}_{\mathit{ev\text{-}inc}}(t > 0, \breve{\varphi}, \vec{a}) := t > 0 \wedge t \leq t(\vec{a})} \\
 & & \mathit{if} \models (t \leq t(\vec{a}) \wedge \breve{\varphi}) \implies t(\vec{a}) \leq t(\vec{a}^2(\vec{x})) \\
\textbf{Fixpoint} & \multicolumn{2}{l}{\mathit{accel}_{\mathit{fp}}(t > 0, \breve{\varphi}, \vec{a}) := t > 0 \wedge \bigwedge_{x \in \mathit{closure}_{\vec{a}}(t)} x = x(\vec{a})} \\
 & & \mathit{where}\ \mathit{closure}_{\vec{a}}(t) := \bigcup_{i \in \mathbb{N}} \mathcal{V}\mathit{ars}(t(\vec{a}^i(\vec{x})))
\end{array}
$$

The above five techniques are taken from [10,11], where only deterministic loops are considered (i.e., there are no temporary variables). Lifting them to non-deterministic loops in a way that allows for *exact* conditional acceleration techniques (which capture all possible program runs) is non-trivial and beyond the scope of this paper. Thus, we sacrifice exactness and treat temporary variables like additional constant program variables whose update is the identity, resulting in a sound under-approximation (that captures a subset of all possible runs).

So essentially, **Increase** and **Decrease** handle inequations $t > 0$ in the loop guard where t increases or decreases (weakly) monotonically when applying the loop's update. The canonical examples where **Increase** or **Decrease** applies are

$$f(x, \ldots) \to f(x+1, \ldots)\,[x > 0 \wedge \ldots] \quad \text{or} \quad f(x, \ldots) \to f(x-1, \ldots)\,[x > 0 \wedge \ldots],$$

respectively. **Eventual Decrease** applies if t never increases again once it starts to decrease. The canonical example is $f(x, y, \ldots) \to f(x + y, y - 1, \ldots)\,[x > 0 \wedge \ldots]$. Similarly, **Eventual Increase** applies if t never decreases again once it starts to increase. **Fixpoint** can be used for inequations $t > 0$ that do not behave (eventually) monotonically. It should only be used if $\mathit{accel}_{\mathit{fp}}(t > 0, \breve{\varphi}, \vec{a})$ is satisfiable.

LoAT uses the *acceleration calculus* of [10]. It operates on *acceleration problems* $\llbracket \psi \mid \breve{\varphi} \mid \widehat{\varphi} \rrbracket_{\vec{a}}$, where ψ (which is initially $\top$) is repeatedly refined. When it stops, ψ is used as the guard of the resulting accelerated transition. The formulas $\breve{\varphi}$ and $\widehat{\varphi}$ are the parts of the loop guard that have already or have not yet been handled, respectively. So $\breve{\varphi}$ is initially $\top$, and $\widehat{\varphi}$ and $\vec{a}$ are initialized with the guard φ and the update of the loop $f(\vec{x}) \xrightarrow{p} f(\vec{a})\,[\varphi]$ under consideration, i.e., the initial acceleration problem is $\llbracket \top \mid \top \mid \varphi \rrbracket_{\vec{a}}$. Once $\widehat{\varphi}$ is $\top$, the loop is accelerated to $f(\vec{x}) \xrightarrow{q} f(\vec{a}^n(\vec{x}))\,[\psi \wedge n > 0]$, where the cost q and a closed form for $\vec{a}^n(\vec{x})$ are computed by the recurrence solver PURRS [2].

Definition 3 (Acceleration Calculus for Conjunctive Loops). *The relation $\rightsquigarrow$ on acceleration problems is defined as*

$$\frac{\mathit{accel}(\xi, \breve{\varphi}, \vec{a}) = \psi_2}{\llbracket \psi_1 \mid \breve{\varphi} \mid \xi \wedge \widehat{\varphi} \rrbracket_{\vec{a}} \rightsquigarrow \llbracket \psi_1 \wedge \psi_2 \mid \breve{\varphi} \wedge \xi \mid \widehat{\varphi} \rrbracket_{\vec{a}}} \qquad \begin{array}{l}\mathit{accel\ is\ a\ conditional} \\ \mathit{acceleration\ technique}\end{array}$$

So to accelerate a loop, one picks a not yet handled part ξ of the guard in each step. When accelerating $f(\vec{x}) \to f(\vec{a})\,[\xi]$ using a conditional acceleration technique *accel*, one may assume $\forall i \in [0,n).\ \breve{\varphi}(\vec{a}^i(\vec{x}))$. The result of *accel* is conjoined to the result ψ_1 computed so far, and ξ is moved from the third to the second component of the problem, i.e., to the already handled part of the guard.

Example 4 (Acceleration Calculus). We show how to accelerate the loop

$$f(x,y) \xrightarrow{x} f(x-y,y)\,[x>0 \land y \geq 0] \qquad \text{to}$$
$$f(x,y) \xrightarrow{(x+\frac{y}{2})\cdot n - \frac{y}{2}\cdot n^2} f(x-n\cdot y,y)\,[y \geq 0 \land x-(n-1)\cdot y > 0 \land n > 0]\,.$$

The closed form $\vec{a}^n(x) = (x-n\cdot y, y)$ can be computed via recurrence solving. Similarly, the cost $(x+\frac{y}{2})\cdot n - \frac{y}{2}\cdot n^2$ of n loop iterations is obtained by solving the following recurrence relation (where $c^{(n)}$ and $x^{(n)}$ denote the cost and the value of x after n applications of the transition, respectively).

$$c^{(n)} = c^{(n-1)} + x^{(n-1)} = c^{(n-1)} + x - (n-1)\cdot y \qquad \text{and} \qquad c^{(1)} = x.$$

The guard is computed as follows:

$$\begin{aligned}\llbracket \top \mid \top \mid x>0 \land y \geq 0 \rrbracket_{\vec{a}} &\rightsquigarrow \llbracket y \geq 0 \mid y \geq 0 \mid x > 0 \rrbracket_{\vec{a}} \\ &\rightsquigarrow \llbracket y \geq 0 \land x-(n-1)\cdot y > 0 \mid y \geq 0 \land x > 0 \mid \top \rrbracket_{\vec{a}}\,.\end{aligned}$$

In the 1^{st} step, we have $\xi = (y \geq 0)$ and $accel_{inc}(y \geq 0, \top, \vec{a}) = (y \geq 0)$. In the 2^{nd} step, we have $\xi = (x > 0)$ and $accel_{dec}(x > 0, y \geq 0, \vec{a}) = (x-(n-1)\cdot y > 0)$. So the inequation $x-(n-1)\cdot y > 0$ ensures n-invariance of $x > 0$.

5 Efficient Loop Acceleration Using Unsat Cores

Each attempt to apply a conditional acceleration technique other than **Fixpoint** requires proving an implication, which is implemented via SMT solving by proving unsatisfiability of its negation. For **Fixpoint**, satisfiability of $accel_{fp}(t > 0, \breve{\varphi}, \vec{a})$ is checked via SMT. So even though LoAT restricts ξ to atoms, up to $\Theta(m^2)$ attempts to apply a conditional acceleration technique are required to accelerate a loop whose guard contains m inequations using a naive strategy ($5\cdot m$ attempts for the 1^{st} $\rightsquigarrow$-step, $5\cdot(m-1)$ attempts for the 2^{nd} step, $\ldots$).

To improve efficiency, LoAT uses a novel encoding that requires just $5\cdot m$ attempts. For any $\alpha \in AT_{imp} = \{inc, dec, ev\text{-}dec, ev\text{-}inc\}$, let $encode_\alpha(\xi, \breve{\varphi}, \vec{a})$ be the implication that has to be valid in order to apply $accel_\alpha$, whose premise is of the form $\ldots \land \breve{\varphi}$. Instead of repeatedly refining $\breve{\varphi}$, LoAT tries to prove validity[2] of $encode_{\alpha,\xi} := encode_\alpha(\xi, \varphi \setminus \{\xi\}, \vec{a})$ for each $\alpha \in AT_{imp}$ and each $\xi \in \varphi$, where φ is the (conjunctive) guard of the transition that should be accelerated. Again,

[2] Here and in the following, we unify conjunctions of atoms with sets of atoms.

proving validity of an implication is equivalent to proving unsatisfiability of its negation. So if validity of $\mathit{encode}_{\alpha,\xi}$ can be shown, then SMT solvers can also provide an *unsat core* for $\neg\mathit{encode}_{\alpha,\xi}$.

Definition 5 (Unsat Core). *Given a conjunction ψ, we call each unsatisfiable subset of ψ an* unsat core *of ψ.*

Theorem 6 shows that when handling an inequation ξ, one only has to require n-invariance for the elements of $\varphi \setminus \{\xi\}$ that occur in an unsat core of $\neg\mathit{encode}_{\alpha,\xi}$. Thus, an unsat core of $\neg\mathit{encode}_{\alpha,\xi}$ can be used to determine which prerequisites $\breve{\varphi}$ are needed for the inequation ξ. This information can then be used to find a suitable order for handling the inequations of the guard. Thus, in this way one only has to check (un)satisfiability of the $4 \cdot m$ formulas $\neg\mathit{encode}_{\alpha,\xi}$. If no such order is found, then LoAT either fails to accelerate the loop under consideration, or it resorts to using **Fixpoint**, as discussed below.

Theorem 6 (Unsat Core Induces $\rightsquigarrow$-Step). *Let $\mathit{deps}_{\alpha,\xi}$ be the intersection of $\varphi \setminus \{\xi\}$ and an unsat core of $\neg\mathit{encode}_{\alpha,\xi}$. If $\breve{\varphi}$ implies $\mathit{deps}_{\alpha,\xi}$, then $\mathit{accel}_{\alpha}(\xi, \breve{\varphi}, \vec{a}) = \mathit{accel}_{\alpha}(\xi, \varphi \setminus \{\xi\}, \vec{a})$.*

Example 7 (Controlling Acceleration Steps via Unsat Cores). Reconsider Example 4. Here, LoAT would try to prove, among others, the following implications:

$$\mathit{encode}_{\mathit{dec},x>0} \quad = \quad (x - y > 0 \wedge y > 0) \implies x > 0 \tag{1}$$

$$\mathit{encode}_{\mathit{inc},y>0} \quad = \quad (y > 0 \wedge x > 0) \implies y > 0 \tag{2}$$

To do so, it would try to prove unsatisfiability of $\neg\mathit{encode}_{\alpha,\xi}$ via SMT. For (1), we get $\neg\mathit{encode}_{\mathit{dec},x>0} = (x - y > 0 \wedge y > 0 \wedge x \leq 0)$, whose only unsat core is $\neg\mathit{encode}_{\mathit{dec},x>0}$, and its intersection with $\varphi \setminus \{x > 0\} = \{y > 0\}$ is $\{y > 0\}$.

For (2), we get $\neg\mathit{encode}_{\mathit{inc},y>0} = (y > 0 \wedge x > 0 \wedge y \leq 0)$, whose minimal unsat core is $y > 0 \wedge y \leq 0$, and its intersection with $\varphi \setminus \{y > 0\} = \{x > 0\}$ is empty. So by Theorem 6, we have $\mathit{accel}_{\mathit{inc}}(y > 0, \top, \vec{a}) = \mathit{accel}_{\mathit{inc}}(y > 0, x > 0, \vec{a})$.

In this way, validity of $\mathit{encode}_{\alpha_1,x>0}$ and $\mathit{encode}_{\alpha_2,y>0}$ is proven for all $\alpha_1 \in \mathit{AT}_{\mathit{imp}} \setminus \{\mathit{inc}\}$ and all $\alpha_2 \in \mathit{AT}_{\mathit{imp}}$. However, the premise $x \leq x - y \wedge y > 0$ of $\mathit{encode}_{\mathit{ev\text{-}inc},x>0}$ is unsatisfiable and thus a corresponding acceleration step would yield a transition with unsatisfiable guard. To prevent that, LoAT only uses a technique $\alpha \in \mathit{AT}_{\mathit{imp}}$ for ξ if the premise of $\mathit{encode}_{\alpha,\xi}$ is satisfiable.

So for each inequation ξ from φ, LoAT synthesizes up to 4 potential $\rightsquigarrow$-steps corresponding to $\mathit{accel}_{\alpha}(\xi, \mathit{deps}_{\alpha,\xi}, \vec{a})$, where $\alpha \in \mathit{AT}_{\mathit{imp}}$. If validity of $\mathit{encode}_{\alpha,\xi}$ cannot be shown for any $\alpha \in \mathit{AT}_{\mathit{imp}}$, then LoAT tries to prove satisfiability of $\mathit{accel}_{\mathit{fp}}(\xi, \top, \vec{a})$ to see if **Fixpoint** should be applied. Note that the 2^{nd} argument of $\mathit{accel}_{\mathit{fp}}$ is irrelevant, i.e., **Fixpoint** does not benefit from previous acceleration steps and thus $\rightsquigarrow$-steps that use it do not have any dependencies.

It remains to find a suitably ordered subset S of m $\rightsquigarrow$-steps that constitutes a successful $\rightsquigarrow$-sequence. In the following, we define $\mathit{AT} := \mathit{AT}_{\mathit{imp}} \cup \{\mathit{fp}\}$ and we extend the definition of $\mathit{deps}_{\alpha,\xi}$ to the case $\alpha = \mathit{fp}$ by defining $\mathit{deps}_{\mathit{fp},\xi} := \varnothing$.

Lemma 8. *Let $C \subseteq AT \times \varphi$ be the smallest set such that $(\alpha, \xi) \in C$ implies*

(a) if $\alpha \in AT_{imp}$, then $encode_{\alpha,\xi}$ is valid and its premise is satisfiable,
(b) if $\alpha = fp$, then $accel_{fp}(\xi, \top, \vec{a})$ is satisfiable, and
(c) $deps_{\alpha,\xi} \subseteq \{\xi' \mid (\alpha', \xi') \in C \text{ for some } \alpha' \in AT\}$.

Let $S := \{(\alpha, \xi) \in C \mid \alpha \geq_{AT} \alpha' \text{ for all } (\alpha', \xi) \in C\}$ where $>_{AT}$ is the total order inc $>_{AT}$ dec $>_{AT}$ ev-dec $>_{AT}$ ev-inc $>_{AT}$ fp. We define $(\alpha', \xi') \prec (\alpha, \xi)$ if $\xi' \in deps_{\alpha,\xi}$. Then $\prec$ is a strict (and hence, well-founded) order on S.

The order $>_{AT}$ in Lemma 8 corresponds to the order proposed in [10]. Note that the set C can be computed without further (potentially expensive) SMT queries by a straightforward fixpoint iteration, and well-foundedness of $\prec$ follows from minimality of C. For Example 7, we get

$$C = \{(dec, x > 0), (ev\text{-}dec, x > 0)\} \cup \{(\alpha, y > 0) \mid \alpha \in AT\} \qquad \text{and}$$
$$S = \{(dec, x > 0), (inc, y > 0)\} \text{ with } (inc, y > 0) \prec (dec, x > 0).$$

Finally, we can construct a valid $\rightsquigarrow$-sequence via the following theorem.

Theorem 9. (Finding $\rightsquigarrow$-Sequences). *Let S be defined as in Lemma 8 and assume that for each $\xi \in \varphi$, there is an $\alpha \in AT$ such that $(\alpha, \xi) \in S$. W.l.o.g., let $\varphi = \bigwedge_{i=1}^{m} \xi_i$ where $(\alpha_1, \xi_1) \prec' \ldots \prec' (\alpha_m, \xi_m)$ for some strict total order $\prec'$ containing $\prec$, and let $\breve{\varphi}_j := \bigwedge_{i=1}^{j} \xi_i$. Then for all $j \in [0, m)$, we have:*

$$\left\llbracket \bigwedge_{i=1}^{j} accel_{\alpha_i}(\xi_i, \breve{\varphi}_{i-1}, \vec{a}) \;\middle|\; \breve{\varphi}_j \;\middle|\; \bigwedge_{i=j+1}^{m} \xi_i \right\rrbracket_{\vec{a}} \rightsquigarrow \left\llbracket \bigwedge_{i=1}^{j+1} accel_{\alpha_i}(\xi_i, \breve{\varphi}_{i-1}, \vec{a}) \;\middle|\; \breve{\varphi}_{j+1} \;\middle|\; \bigwedge_{i=j+2}^{m} \xi_i \right\rrbracket_{\vec{a}}$$

In our example, we have $\prec' = \prec$ as $\prec$ is total. Thus, we obtain a $\rightsquigarrow$-sequence by first processing $y > 0$ with **Increase** and then processing $x > 0$ with **Decrease**.

6 Proving Non-Termination of Simple Loops

To prove non-termination, LoAT uses a variation of the calculus from Sect. 4, see [11]. To adapt it for proving non-termination, further restrictions have to be imposed on the conditional acceleration techniques, resulting in the notion of *conditional non-termination techniques*, see [11, Def. 10]. We denote a $\rightsquigarrow$-step that uses a conditional non-termination technique with $\rightsquigarrow_{nt}$.

Theorem 10. (Proving Non-Termination via $\rightsquigarrow_{nt}$). *Let $f(\vec{x}) \to f(\vec{a})\,[\varphi] \in \mathcal{T}$. If $\llbracket \top \mid \top \mid \varphi \rrbracket_{\vec{a}} \rightsquigarrow_{nt}^{*} \llbracket \psi \mid \varphi \mid \top \rrbracket_{\vec{a}}$, then for every $\vec{c} \in \mathbb{Z}^d$ where $\psi(\vec{c})$ is satisfiable, the configuration $f(\vec{c})$ admits an infinite $\to_{\mathcal{T}}$-sequence.*

The conditional non-termination techniques used by LoAT are **Increase**, **Eventual Increase**, and **Fixpoint**. So non-termination proofs can be synthesized while trying to accelerate a loop with very little overhead. After successfully accelerating a loop as explained in Sect. 5, LoAT tries to find a second suitably ordered $\rightsquigarrow$-sequence, where it only considers the conditional non-termination techniques mentioned above. If LoAT succeeds, then it has found a $\rightsquigarrow_{nt}$-sequence which gives rise to a proof of non-termination via Theorem 10.

7 Implementation, Experiments, and Conclusion

Our implementation in LoAT can parse three widely used formats for ITSs (see [13]), and it is configurable via a minimalistic set of command-line options:

`--timeout` to set a timeout in seconds
`--proof-level` to set the verbosity of the proof output
`--plain` to switch from colored to monochrome proof-output
`--limit-strategy` to choose a strategy for solving limit problems, see [9]
`--mode` to choose an analysis mode for LoAT (`complexity` or `non_termination`)

We evaluate three versions of LoAT: LoAT '19 uses templates to find invariants that facilitate loop acceleration for proving non-termination [8]; LoAT '20 deduces worst-case lower bounds based on loop acceleration via *metering functions* [9]; and LoAT '22 applies the calculus from [10,11] as described in Sect. 5 and 6. We also include three other state-of-the-art termination tools in our evaluation: T2 [6], VeryMax [16], and iRankFinder [3,7]. Regarding complexity, the only other tool for worst-case lower bounds of ITSs is LOBER [1]. However, we do not compare with LOBER, as it only analyses (multi-path) loops instead of full ITSs.

We use the examples from the categories *Termination* (1222 examples) and *Complexity of ITSs* (781 examples), respectively, of the *Termination Problems Data Base* [19]. All benchmarks have been performed on *StarExec* [18] (Intel Xeon E5-2609, 2.40GHz, 264GB RAM [17]) with a wall clock timeout of 300 s.

	No	Yes	Avg. Rt	Median Rt	Std. Dev. Rt
LoAT '22	493	0	9.4	0.2	41.5
LoAT '19	459	0	22.6	1.5	67.5
T2	438	610	22.6	1.2	66.7
VeryMax	419	628	29.9	1.0	66.7
iRankFinder	399	634	44.1	4.9	89.1

		LoAT '22					
	$rc_{\mathcal{T}}(n)$	$\Omega(1)$	$\Omega(n)$	$\Omega(n^2)$	$\Omega(n^{>2})$	EXP	$\Omega(\omega)$
LoAT '20	$\Omega(1)$	180	63	1	–	–	12
	$\Omega(n)$	6	218	3	–	–	–
	$\Omega(n^2)$	–	1	69	–	–	–
	$\Omega(n^{>2})$	–	–	–	7	–	–
	EXP	1	–	–	–	4	–
	$\Omega(\omega)$	–	–	–	–	–	216

By the table on the left, LoAT '22 is the most powerful tool for non-termination. The improvement over LoAT '19 demonstrates that the calculus from [10,11] is more powerful and efficient than the approach from [8]. The last three columns show the average, the median, and the standard deviation of the wall clock runtime, including examples where the timeout was reached.

The table on the right shows the results for complexity. The diagonal corresponds to examples where LoAT '20 and LoAT '22 yield the same result. The entries above or below the diagonal correspond to examples where LoAT '22 or LoAT '20 is better, respectively. There are 8 regressions and 79 improvements, so the calculus from [10,11] used by LoAT '22 is also beneficial for lower bounds.

LoAT is open source and its source code is available on GitHub [12]. See [13,14] for details on our evaluation, related work, all proofs, and a pre-compiled binary.

References

1. Albert, E., Genaim, S., Martin-Martin, E., Merayo, A., Rubio, A.: Lower-bound synthesis using loop specialization and Max-SMT. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12760, pp. 863–886. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81688-9_40
2. Bagnara, R., Pescetti, A., Zaccagnini, A., Zaffanella, E.: PURRS: towards computer algebra support for fully automatic worst-case complexity analysis. CoRR abs/cs/0512056 (2005). https://arxiv.org/abs/cs/0512056
3. Ben-Amram, A.M., Doménech, J.J., Genaim, S.: Multiphase-linear ranking functions and their relation to recurrent sets. In: Chang, B.-Y.E. (ed.) SAS 2019. LNCS, vol. 11822, pp. 459–480. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32304-2_22
4. Bozga, M., Gîrlea, C., Iosif, R.: Iterating octagons. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 337–351. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00768-2_29
5. Bozga, M., Iosif, R., Konečný, F.: Fast acceleration of ultimately periodic relations. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 227–242. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_23
6. Brockschmidt, M., Cook, B., Ishtiaq, S., Khlaaf, H., Piterman, N.: T2: temporal property verification. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 387–393. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_22
7. Doménech, J.J., Genaim, S.: iRankFinder. In: Lucas, S. (ed.) WST 2018, p. 83 (2018). http://wst2018.webs.upv.es/wst2018proceedings.pdf
8. Frohn, F., Giesl, J.: Proving non-termination via loop acceleration. In: Barrett, C.W., Yang, J. (eds.) FMCAD 2019, pp. 221–230 (2019). https://doi.org/10.23919/FMCAD.2019.8894271
9. Frohn, F., Naaf, M., Brockschmidt, M., Giesl, J.: Inferring lower runtime bounds for integer programs. ACM TOPLAS **42**(3), 13:1–13:50 (2020). https://doi.org/10.1145/3410331. Revised and extended version of a paper which appeared in IJCAR 2016, pp. 550–567. LNCS, vol. 9706 (2016)
10. Frohn, F.: A calculus for modular loop acceleration. In: Biere, A., Parker, D. (eds.) TACAS 2020. LNCS, vol. 12078, pp. 58–76. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45190-5_4
11. Frohn, F., Fuhs, C.: A calculus for modular loop acceleration and non-termination proofs. CoRR abs/2111.13952 (2021). https://arxiv.org/abs/2111.13952, to appear in STTT
12. Frohn, F.: LoAT on GitHub. https://github.com/aprove-developers/LoAT
13. Frohn, F., Giesl, J.: Empirical evaluation of: proving non-termination and lower runtime bounds with LoAT. https://ffrohn.github.io/loat-tool-paper-evaluation
14. Frohn, F., Giesl, J.: Proving non-termination and lower runtime bounds with LoAT (System Description). CoRR abs/2202.04546 (2022). https://arxiv.org/abs/2202.04546
15. Kroening, D., Lewis, M., Weissenbacher, G.: Under-approximating loops in C programs for fast counterexample detection. Formal Methods Syst. Des. **47**(1), 75–92 (2015). https://doi.org/10.1007/s10703-015-0228-1
16. Larraz, D., Nimkar, K., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving non-termination using Max-SMT. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 779–796. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_52

17. StarExec hardware specifications. https://www.starexec.org/starexec/public/machine-specs.txt
18. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: a cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 367–373. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_28
19. Termination Problems Data Base (TPDB, Git SHA 755775). https://github.com/TermCOMP/TPDB

Implicit Definitions with Differential Equations for KeYmaera X
(System Description)

James Gallicchio(✉), Yong Kiam Tan(✉), Stefan Mitsch(✉), and André Platzer(✉)

Computer Science Department, Carnegie Mellon University, Pittsburgh, USA
jgallicc@andrew.cmu.edu, {yongkiat,smitsch,aplatzer}@cs.cmu.edu

Abstract. Definition packages in theorem provers provide users with means of defining and organizing concepts of interest. This system description presents a new definition package for the hybrid systems theorem prover KeYmaera X based on differential dynamic logic (dL). The package adds KeYmaera X support for user-defined smooth functions whose graphs can be implicitly characterized by dL formulas. Notably, this makes it possible to implicitly characterize functions, such as the exponential and trigonometric functions, as solutions of differential equations and then prove properties of those functions using dL's differential equation reasoning principles. Trustworthiness of the package is achieved by minimally extending KeYmaera X's soundness-critical kernel with a single axiom scheme that expands function occurrences with their implicit characterization. Users are provided with a high-level interface for defining functions and non-soundness-critical tactics that automate low-level reasoning over implicit characterizations in hybrid system proofs.

Keywords: Definitions · Differential dynamic logic · Verification of hybrid systems · Theorem proving

1 Introduction

KeYmaera X [7] is a theorem prover implementing differential dynamic logic dL [17,19–21] for specifying and verifying properties of hybrid systems mixing discrete dynamics and differential equations. Definitions enable users to express complex theorem statements in concise terms, e.g., by modularizing hybrid system models and their proofs [14]. Prior to this work, KeYmaera X had only one mechanism for definition, namely, non-recursive abbreviations via uniform substitution [14,20]. This restriction meant that common and useful functions, e.g., the trigonometric and exponential functions, could not be directly used in KeYmaera X, even though they can be uniquely characterized by dL formulas [17].

This system description introduces a new KeYmaera X definitional mechanism where functions are *implicitly defined* in dL as solutions of ordinary differential equations (ODEs). Although definition packages are available in most

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 723–733, 2022.
https://doi.org/10.1007/978-3-031-10769-6_42

general-purpose proof assistants, our package is novel in tackling the question of how best to support user-defined functions in the *domain-specific* setting for hybrid systems. In contrast to tools with builtin support for *some* fixed subsets of special functions [1,9,23]; or higher-order logics that can work with functions via their infinitary series expansions [4], e.g., $\exp(t) = \sum_{i=0}^{\infty} \frac{t^i}{i!}$; our package strikes a balance between practicality and generality by allowing users to define and reason about *any* function characterizable in dL as the solution of an ODE (Sect. 2), e.g., $\exp(t)$ solves the ODE $e' = e$ with initial value $e(0) = 1$.

Theoretically, implicit definitions strictly expand the class of ODE invariants amenable to dL's complete ODE invariance proof principles [22]; such invariants play a key role in ODE safety proofs [21] (see Proposition 3). In practice, arithmetical identities and other specifications involving user-defined functions are proved by automatically unfolding their implicit ODE characterizations and reusing existing KeYmaera X support for ODE reasoning (Sect. 3). The package is designed to provide seamless integration of implicit definitions in KeYmaera X and its usability is demonstrated on several hybrid system verification examples drawn from the literature that involve special functions (Sect. 4).

All proofs are in the supplement [8]. The definitions package is part of KeYmaera X with a usage guide at: http://keymaeraX.org/keymaeraXfunc/.

2 Interpreted Functions in Differential Dynamic Logic

This section briefly recalls differential dynamic logic (dL) [17,18,20,21] and explains how its term language is extended to support implicit function definitions.

Syntax. Terms $e, \tilde{e}$ and formulas ϕ, ψ in dL are generated by the following grammar, with variable x, rational constant c, k-ary function symbols h (for any $k \in \mathbb{N}$), comparison operator $\sim \in \{=, \neq, \geq, >, \leq, <\}$, and hybrid program α:

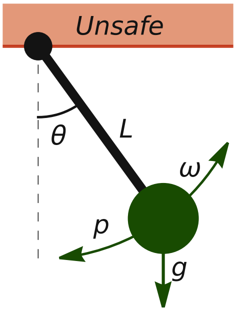
$$e, \tilde{e} ::= x \mid c \mid e + \tilde{e} \mid e \cdot \tilde{e} \mid h(e_1, \ldots, e_k) \tag{1}$$

$$\phi, \psi ::= e \sim \tilde{e} \mid \phi \land \psi \mid \phi \lor \psi \mid \neg\phi \mid \forall x \phi \mid \exists x \phi \mid [\alpha]\,\phi \mid \langle\alpha\rangle\,\phi \tag{2}$$

The terms and formulas above extend the first-order language of real arithmetic ($\text{FOL}_{\mathbb{R}}$) with the box ($[\alpha]\,\phi$) and diamond ($\langle\alpha\rangle\,\phi$) modality formulas which express that *all* or *some* runs of hybrid program α satisfy postcondition ϕ, respectively. Table 1 gives an intuitive overview of dL's hybrid programs language for modeling systems featuring discrete and continuous dynamics and their interactions thereof. In dL's uniform substitution calculus, function symbols h are *uninterpreted*, i.e., they semantically correspond to an arbitrary (smooth) function. Such uninterpreted function symbols (along with uninterpreted predicate and program symbols) are crucially used to give a parsimonious axiomatization of dL based on uniform substitution [20] which, in turn, enables a trustworthy microkernel implementation of the logic in the theorem prover KeYmaera X [7,16].

Table 1. Syntax and informal semantics of hybrid programs

Program	Behavior
$?\phi$	Stay in the current state if ϕ is true, otherwise abort and discard run
$x := e$	Store the value of term e in variable x
$x := *$	Store an arbitrary real value in variable x
$x' = f(x) \,\&\, Q$	Continuously follow ODE $x' = f(x)$ in domain Q for any duration ≥ 0
$\texttt{if}(\phi)\,\alpha$	Run program α if ϕ is true, otherwise skip. Definable by $?\phi; \alpha \cup ?\neg\phi$
$\alpha; \beta$	Run program α, then run program β in any resulting state(s)
$\alpha \cup \beta$	Nondeterministically run either program α or program β
α^*	Nondeterministically repeat program α for n iterations, for any $n \in \mathbb{N}$
$\{\alpha\}$	For readability, braces are used to group and delimit hybrid programs



Hybrid program model (auxiliary variables s, c):

$$\hat{\alpha}_s \equiv \left(\begin{array}{l} s := *; c := *; ?\phi_{\sin}(s,\theta); ?\phi_{\cos}(c,\theta); \\ p := *; \texttt{if}\left(\frac{1}{2}(\omega - p)^2 < \frac{g}{L}c\right) \{\omega := \omega - p\}; \\ \{\theta' = \omega, \omega' = -\frac{g}{L}s - k\omega, s' = \omega c, c' = -\omega s\} \end{array} \right)^*$$

Hybrid program model (trigonometric functions):

$$\alpha_s \equiv \left(\begin{array}{l} p := *; \texttt{if}\left(\frac{1}{2}(\omega - p)^2 < \frac{g}{L}\cos(\theta)\right) \{\omega := \omega - p\}; \\ \{\theta' = \omega, \omega' = -\frac{g}{L}\sin(\theta) - k\omega\} \end{array} \right)^*$$

dL safety specification:

$$\phi_s \equiv g > 0 \land L > 0 \land k > 0 \land \theta = 0 \land \omega = 0 \rightarrow [\alpha_s]\, |\theta| < \frac{\pi}{2}$$

Fig. 1. Running example of a swinging pendulum driven by an external force (left), its hybrid program models and dL safety specification (right). Program α_s uses trigonometric functions directly, while program $\hat{\alpha}_s$ uses variables s, c to implicitly track the values of $\sin(\theta)$ and $\cos(\theta)$, respectively (additions in red). The implicit characterizations $\phi_{\sin}(s,\theta), \phi_{\cos}(c,\theta)$ are defined in (4), (5) and are not repeated here for brevity. (Color figure online)

Running Example. Adequate modeling of hybrid systems often requires the use of *interpreted* function symbols that denote specific functions of interest. As a running example, consider the swinging pendulum shown in Fig. 1. The ODEs describing its continuous motion are $\theta' = \omega, \omega' = -\frac{g}{L}\sin(\theta) - k\omega$, where θ is the swing angle, ω is the angular velocity, and g, k, L are the gravitational constant, coefficient of friction, and length of the rigid rod suspending the pendulum, respectively. The hybrid program α_s models an external force that repeatedly pushes the pendulum and changes its angular velocity by a nondeterministically chosen value p; the guard `if(...)` condition is designed to ensure that the push does not cause the pendulum to swing above the horizontal as specified by ϕ_s. Importantly, the function symbols $\sin, \cos$ must denote the usual real trigonometric functions in α_s. Program $\hat{\alpha}_s$ shows the same pendulum modeled in dL *without* the use of interpreted symbols, but instead using auxiliary variables s, c. Note that $\hat{\alpha}_s$ is cumbersome and subtle to get right: the implicit characterizations $\phi_{\sin}(s, \theta), \phi_{\cos}(c, \theta)$ from (4), (5) are lengthy and the differential equations $s' = \omega c, c' = -\omega s$ must be manually calculated and added to ensure that s, c correctly track the trigonometric functions as θ evolves continuously [18,22].

Interpreted Functions. To enable extensible use of interpreted functions in dL, the term grammar (1) is enriched with k-ary function symbols h that carry an *interpretation* annotation [5,27], $h_{\ll\phi\gg}$, where $\phi \equiv \phi(x_0, y_1, \dots, y_k)$ is a dL formula with free variables in $x_0, y_1, \dots, y_k$ and no uninterpreted symbols. Intuitively, ϕ is a formula that characterizes the graph of the intended interpretation for h, where $y_1, \dots, y_k$ are inputs to the function and x_0 is the output. Since ϕ depends only on the values of its free variables, its formula semantics $[\![\phi]\!]$ can be equivalently viewed as a subset of Euclidean space $[\![\phi]\!] \subseteq \mathbb{R} \times \mathbb{R}^k$ [20,21]. The dL term semantics $\nu[\![e]\!]$ [20,21] in a state ν is extended with a case for terms $h_{\ll\phi\gg}(e_1, \dots, e_k)$ by evaluation of the smooth C^∞ function characterized by $[\![\phi]\!]$:

$$\nu[\![h_{\ll\phi\gg}(e_1, \dots, e_k)]\!] = \begin{cases} \hat{h}(\nu[\![e_1]\!], \dots, \nu[\![e_k]\!]) & \text{if } [\![\phi]\!] \text{ graph of smooth } \hat{h}{:}\mathbb{R}^k{\to}\mathbb{R} \\ 0 & \text{otherwise} \end{cases}$$

This semantics says that, if the relation $[\![\phi]\!] \subseteq \mathbb{R} \times \mathbb{R}^k$ is the graph of some smooth C^∞ function $\hat{h} : \mathbb{R}^k \to \mathbb{R}$, then the annotated syntactic symbol $h_{\ll\phi\gg}$ is interpreted semantically as $\hat{h}$. Note that the graph relation uniquely defines $\hat{h}$ (if it exists). Otherwise, $h_{\ll\phi\gg}$ is interpreted as the constant zero function which ensures that the term semantics remain well-defined for all terms. An alternative is to leave the semantics of some terms (possibly) undefined, but this would require more extensive changes to the semantics of dL and extra case distinctions during proofs [2].

Axiomatics and Differentially-Defined Functions. To support reasoning for implicit definitions, annotated interpretations are reified to characterization axioms for expanding interpreted functions in the following lemma.

Lemma 1. (Function interpretation). *The FI axiom (below) for dL is sound where h is a k-ary function symbol and the formula semantics $[\![\phi]\!]$ is the graph of a smooth C^∞ function $\hat{h} : \mathbb{R}^k \to \mathbb{R}$.*

$$\text{FI} \quad e_0 = h_{\ll\phi\gg}(e_1, \dots, e_k) \leftrightarrow \phi(e_0, e_1, \dots, e_k)$$

Axiom FI enables reasoning for terms $h_{\ll\phi\gg}(e_1, \dots, e_k)$ through their implicit interpretation ϕ, but Lemma 1 does not directly yield an implementation because it has a soundness-critical side condition that interpretation ϕ characterizes the graph of a smooth C^∞ function. It is possible to syntactically characterize this side condition [2], e.g., the formula $\forall y_1, \ldots, y_k \exists x_0 \phi(x_0, y_1, \ldots, y_k)$ expresses that the graph represented by ϕ has at least one output value x_0 for each input value $y_1, \ldots, y_k$, but this burdens users with the task of proving this side condition in dL before working with their desired function. The KeYmaera X definition package opts for a middle ground between generality and ease-of-use by implementing FI for univariate, *differentially-defined* functions, i.e., the interpretation ϕ has the following shape, where $x = (x_0, x_1, \ldots, x_n)$ abbreviates a vector of variables, there is one input $t = y_1$, and $X = (X_0, X_1, \ldots, X_n)$, T are dL terms that do not mention any free variables, e.g., are rational constants, which have constant value in any dL state:

$$\phi(x_0, t) \equiv \left\langle x_1, \ldots, x_n := *; \left\{ \begin{array}{l} x' = -f(x,t), t' = -1 \cup \\ x' = f(x,t), t' = 1 \end{array} \right\} \right\rangle \left(\begin{array}{c} x = X \wedge \\ t = T \end{array} \right) \tag{3}$$

Formula (3) says from point x_0, there exists a choice of the remaining coordinates $x_1, \ldots, x_n$ such that it is possible to follow the defining ODE either forward $x' = f(x,t), t' = 1$ or backward $x' = -f(x,t), t' = -1$ in time to reach the initial values $x = X$ at time $t = T$. In other words, the implicitly defined function $h_{\ll \phi(x_0,t) \gg}$ is the x_0-coordinate projected solution of the ODE starting from initial values X at initial time T. For example, the trigonometric functions used in Fig. 1 are differentially-definable as respective projections:

$$\phi_{\sin}(s, t) \equiv \left\langle c := *; \left\{ \begin{array}{l} s' = -c, c' = \quad s, t' = -1 \cup \\ s' = \quad c, c' = -s, t' = \quad 1 \end{array} \right\} \right\rangle \left(\begin{array}{l} s = 0 \wedge c = 1 \wedge \\ t = 0 \end{array} \right) \tag{4}$$

$$\phi_{\cos}(c, t) \equiv \left\langle s := *; \left\{ \begin{array}{l} s' = -c, c' = \quad s, t' = -1 \cup \\ s' = \quad c, c' = -s, t' = \quad 1 \end{array} \right\} \right\rangle \left(\begin{array}{l} s = 0 \wedge c = 1 \wedge \\ t = 0 \end{array} \right) \tag{5}$$

By Picard-Lindelöf [21, Thm. 2.2], the ODE $x' = f(x,t)$ has a unique solution $\Phi : (a, b) \to \mathbb{R}^{n+1}$ on an open interval (a, b) for some $-\infty \leq a < b \leq \infty$. Moreover, $\Phi(t)$ is C^∞ smooth in t because the ODE right-hand sides are dL terms with smooth interpretations [20]. Therefore, the side condition for Lemma 1 reduces to showing that Φ exists globally, i.e., it is defined on $t \in (-\infty, \infty)$.

Lemma 2. (Smooth interpretation). *If formula $\exists x_0 \phi(x_0, t)$ is valid, $\phi(x_0, t)$ from (3) characterizes a smooth C^∞ function and axiom FI is sound for $\phi(x_0, t)$.*

Lemma 2 enables an implementation of axiom FI in KeYmaera X that combines a syntactic check (the interpretation has the shape of formula (3)) and a side condition check (requiring users to prove existence for their interpretations).

The addition of differentially-defined functions to dL strictly increases the deductive power of ODE invariants, a key tool in deductive ODE safety reasoning [21]. Intuitively, the added functions allow direct, syntactic descriptions of invariants, e.g., the exponential or trigonometric functions, that have effective invariance proofs using dL's complete ODE invariance reasoning principles [22].

Proposition 3. (Invariant expressivity). *There are valid polynomial* `dL` *differential equation safety properties which are provable using differentially-defined function invariants but are not provable using polynomial invariants.*

3 KeYmaera X Implementation

The implicit definition package adds interpretation annotations and axiom FI based on Lemma 2 in ≈170 lines of code extensions to KeYmaera X's soundness-critical core [7,16]. This section focuses on non-soundness-critical usability features provided by the package that build on those core changes.

3.1 Core-Adjacent Changes

KeYmaera X has a browser-based user interface with concrete, ASCII-based `dL` syntax [14]. The package extends KeYmaera X's parsers and pretty printers with support for interpretation annotations `h«...»(...)` and users can simultaneously define a family of functions as respective coordinate projections of the solution of an n-dimensional ODE (given initial conditions) with sugared syntax:

```
implicit Real h1(Real t), ..., hn(Real t) = {{initcond};{ODE}}
```

For example, the implicit definitions (4), (5) can be written with the following sugared syntax; KeYmaera X automatically inserts the associated interpretation annotations for the trigonometric function symbols, see the supplement [8] for a KeYmaera X snippet of formula ϕ_s from Fig. 1 using this sugared definition.

```
implicit Real sin(Real t), cos(Real t)
   = {{sin:=0; cos:=1;}; {sin'=cos, cos'=-sin}}
```

In fact, the functions sin, cos, exp are so ubiquitous in hybrid system models that the package builds their definitions in automatically without requiring users to write them explicitly. In addition, although arithmetic involving those functions is undecidable [11,24], KeYmaera X can export those functions whenever its external arithmetic tools have partial arithmetic support for those functions.

3.2 Intermediate and User-Level Proof Automation

The package automatically proves three important lemmas about user-defined functions that can be transparently re-used in all subsequent proofs:

1. It proves the side condition of axiom FI using KeYmaera X's automation for proving sufficient duration existence of solutions for ODEs [26] which automatically shows global existence of solutions for all affine ODEs and some univariate nonlinear ODEs. As an example of the latter, the hyperbolic tanh function is differentially-defined as the solution of ODE $x' = 1 - x^2$ with initial value $x = 0$ at $t = 0$ whose global existence is proved automatically.

2. It proves that the functions have initial values as specified by their interpretation, e.g., $\sin(0) = 0$, $\cos(0) = 1$, and $\tanh(0) = 0$.
3. It proves the *differential axiom* [20] for each function that is used to enable syntactic derivative calculations in dL, e.g., the differential axioms for $\sin, \cos$ are $(\sin(e))' = \cos(e)(e)'$ and $(\cos(e))' = -\sin(e)(e)'$, respectively. Briefly, these axioms are automatically derived in a correct-by-construction manner using dL's syntactic version of the chain rule for differentials [20, Fig. 3], so the rate of change of $\sin(e)$ is the rate of change of $\sin(\cdot)$ with respect to its argument e, multiplied by the rate of change of its argument $(e)'$.

These lemmas enable the use of differentially-defined functions with all existing ODE automation in KeYmaera X [22,26]. In particular, since differentially-defined functions are univariate Noetherian functions, they admit complete ODE invariance reasoning principles in dL [22] as implemented in KeYmaera X.

The package also adds specialized support for arithmetical reasoning over differential definitions to supplement external arithmetic tools in proofs. First, it allows users to manually prove identities and bounds using KeYmaera X's ODE reasoning. For example, the bound $\tanh(\lambda x)^2 < 1$ used in the example α_n from Sect. 4 is proved by *differential unfolding* as follows (see supplement [8]):

$$\frac{\vdash \tanh(0)^2 < 1 \quad \tanh(\lambda v)^2{<}1 \vdash [\{v' = 1 \,\&\, v \le x\} \cup \{v' = -1 \,\&\, v \ge x\}]\, \tanh(\lambda v)^2{<}1}{\vdash \tanh(\lambda x)^2 < 1}$$

This deduction step says that, to show the conclusion (below rule bar), it suffices to prove the premises (above rule bar), i.e., the bound is true at $v = 0$ (left premise) and it is preserved as v is evolved forward $v' = 1$ or backward $v' = -1$ along the real line until it reaches x (right premise). The left premise is proved using the initial value lemma for tanh while the right premise is proved by ODE invariance reasoning with the differential axiom for tanh [22].

Second, the package uses KeYmaera X's uniform substitution mechanism [20] to implement (untrusted) abstraction of functions with fresh variables when solving arithmetic subgoals, e.g., the following arithmetic bound for example α_n is proved by abstraction after adding the bounds $\tanh(\lambda x)^2 < 1, \tanh(\lambda y)^2 < 1$.

$$\begin{aligned}
\textbf{Bound:}\quad & x(\tanh(\lambda x) - \tanh(\lambda y)) + y(\tanh(\lambda x) + \tanh(\lambda y)) \le 2\sqrt{x^2 + y^2} \\
\textbf{Abstracted:}\quad & t_x^2 < 1 \land t_y^2 < 1 \rightarrow x(t_x - t_y) + y(t_x + t_y) \le 2\sqrt{x^2 + y^2}
\end{aligned}$$

4 Examples

The definition package enables users to work with differentially-defined functions in KeYmaera X, including modeling and expressing their design intuitions in proofs. This section applies the package to verify various continuous and hybrid system examples from the literature featuring such functions.

Discretely Driven Pendulum. The specification ϕ_s from Fig. 1 contains a discrete loop whose safety property is proved by a loop invariant, i.e., a formula that is preserved by the discrete and continuous dynamics in each loop iteration [21].

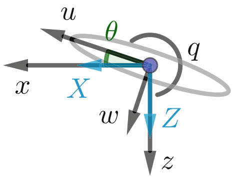
The key invariant is $Inv \equiv \frac{g}{L}(1-\cos\theta) + \frac{1}{2}\omega^2 < \frac{g}{L}$, which expresses that the total energy of the system (sum of potential and kinetic energy on the LHS) is less than the energy needed to cross the horizontal (RHS). The main steps are as follows (proofs for these steps are automated by KeYmaera X):

1. $Inv \rightarrow \left[\texttt{if}\left(\frac{1}{2}(\omega-p)^2 < \frac{g}{L}\cos(\theta)\right)\{\omega := \omega - p\}\right] Inv$, which shows that the discrete guard only allows push p if it preserves the energy invariant, and
2. $Inv \rightarrow \left[\{\theta' = \omega, \omega' = -\frac{g}{L}\sin(\theta) - k\omega\}\right] Inv$, which shows that Inv is an energy invariant of the pendulum's ODE.

Neuron Interaction. The ODE α_n models the interaction between a pair of neurons [12]; its specification ϕ_n nests dL's diamond and box modalities to express that the system norm ($\sqrt{x^2+y^2}$) is asymptotically bounded by 2τ.

$$\alpha_n \equiv x' = -\frac{x}{\tau} + \tanh(\lambda x) - \tanh(\lambda y), y' = -\frac{y}{\tau} + \tanh(\lambda x) + \tanh(\lambda y)$$
$$\phi_n \equiv \tau > 0 \rightarrow \forall \varepsilon{>}0 \langle\alpha_n\rangle\,[\alpha_n]\,\sqrt{x^2+y^2} \leq 2\tau + \varepsilon$$

The verification of ϕ_n uses differentially-defined functions in concert with KeYmaera X's symbolic ODE safety and liveness reasoning [26]. The proof uses a decaying exponential bound $\sqrt{x^2+y^2} \leq \exp(-\frac{t}{\tau})\sqrt{x_0^2+y_0^2} + 2\tau(1-\exp(-\frac{t}{\tau}))$, where the constants x_0, y_0 are symbolic initial values for x, y at initial time $t=0$, respectively. Notably, the arithmetic subgoals from this example are all proved using abstraction and differential unfolding (Sect. 3) without relying on external arithmetic solver support for tanh.



Longitudinal Flight Dynamics. The differential equations α_a below describe the 6th order longitudinal motion of an airplane while climbing or descending [10,25]. The airplane adjusts its pitch angle θ with pitch rate q, which determines its axial velocity u and vertical velocity w, and, in turn, range x and altitude z (illustrated on the right). The physical parameters are: gravity g, mass m, aerodynamic thrust and moment M along the lateral axis, aerodynamic and thrust forces X, Z along x and z, respectively, and the moment of inertia I_{yy}, see [10, Sect. 6.2].

$$\alpha_a \equiv u' = \frac{X}{m} - g\sin(\theta) - qw, \qquad w' = \frac{Z}{m} + g\cos(\theta) + qu, \qquad q' = \frac{M}{I_{yy}},$$
$$x' = \cos(\theta)u + \sin(\theta)w, \qquad z' = -\sin(\theta)u + \cos(\theta)w, \qquad \theta' = q$$

The verification of specification $J \rightarrow [\alpha_a]J$ shows that the safety envelope $J \equiv J_1 \wedge J_2 \wedge J_3$ is invariant along the flow of α_a with algebraic invariants J_i:

$$J_1 \equiv \frac{Mz}{I_{yy}} + g\theta + \left(\frac{X}{m} - qw\right)\cos(\theta) + \left(\frac{Z}{m} + qu\right)\sin(\theta) = 0$$
$$J_2 \equiv \frac{Mz}{I_{yy}} - \left(\frac{Z}{m} + qu\right)\cos(\theta) + \left(\frac{X}{m} - qw\right)\sin(\theta) = 0 \quad J_3 \equiv -q^2 + \frac{2M\theta}{I_{yy}} = 0$$

Additional examples are available in the supplement [8], including: a bouncing ball on a sinusoidal surface [6,13] and a robot collision avoidance model [15].

5 Conclusion

This work presents a convenient mechanism for extending the dL term language with differentially-defined functions, thereby furthering the class of real-world systems amenable to modeling and formalization in KeYmaera X. Minimal soundness-critical changes are made to the KeYmaera X kernel, which maintains its trustworthiness while allowing the use of newly defined functions in concert with all existing dL hybrid systems reasoning principles implemented in KeYmaera X. Future work could formally verify these kernel changes by extending the existing formalization of dL [3]. Further integration of external arithmetic tools [1,9,23] will also help to broaden the classes of arithmetic sub-problems that can be solved effectively in hybrid systems proofs.

Acknowledgments. We thank the anonymous reviewers for their helpful feedback on this paper. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1739629. This research was sponsored by the AFOSR under grant number FA9550-16-1-0288.

References

1. Akbarpour, B., Paulson, L.C.: MetiTarski: an automatic theorem prover for real-valued special functions. J. Autom. Reason. **44**(3), 175–205 (2010). https://doi.org/10.1007/s10817-009-9149-2
2. Bohrer, B., Fernández, M., Platzer, A.: dL_ι: definite descriptions in differential dynamic logic. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 94–110. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_6
3. Bohrer, R., Rahli, V., Vukotic, I., Völp, M., Platzer, A.: Formally verified differential dynamic logic. In: Bertot, Y., Vafeiadis, V. (eds.) CPP, pp. 208–221. ACM (2017). https://doi.org/10.1145/3018610.3018616
4. Boldo, S., Lelay, C., Melquiond, G.: Formalization of real analysis: a survey of proof assistants and libraries. Math. Struct. Comput. Sci. **26**(7), 1196–1233 (2016). https://doi.org/10.1017/S0960129514000437
5. Bonichon, R., Delahaye, D., Doligez, D.: Zenon: an extensible automated theorem prover producing checkable proofs. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 151–165. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75560-9_13
6. Denman, W.: Automated verification of continuous and hybrid dynamical systems. Ph.D. thesis, University of Cambridge, UK (2015)
7. Fulton, N., Mitsch, S., Quesel, J.-D., Völp, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 527–538. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_36
8. Gallicchio, J., Tan, Y.K., Mitsch, S., Platzer, A.: Implicit definitions with differential equations for KeYmaera X (system description). CoRR abs/2203.01272 (2022). http://arxiv.org/abs/2203.01272
9. Gao, S., Kong, S., Clarke, E.M.: dReal: an SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_14

10. Ghorbal, K., Platzer, A.: Characterizing algebraic invariants by differential radical invariants. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 279–294. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_19
11. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik und Physik **38**(1), 173–198 (1931). https://doi.org/10.1007/BF01700692
12. Khalil, H.K.: Nonlinear Systems. Macmillan, New York (1992)
13. Liu, J., Zhan, N., Zhao, H., Zou, L.: Abstraction of elementary hybrid systems by variable transformation. In: Bjørner, N., de Boer, F. (eds.) FM 2015. LNCS, vol. 9109, pp. 360–377. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19249-9_23
14. Mitsch, S.: Implicit and explicit proof management in KeYmaera X. In: Proença, J., Paskevich, A. (eds.) F-IDE. EPTCS, vol. 338, pp. 53–67 (2021). https://doi.org/10.4204/EPTCS.338.8
15. Mitsch, S., Ghorbal, K., Vogelbacher, D., Platzer, A.: Formal verification of obstacle avoidance and navigation of ground robots. Int. J. Robot. Res. **36**(12), 1312–1340 (2017). https://doi.org/10.1177/0278364917733549
16. Mitsch, S., Platzer, A.: A retrospective on developing hybrid system provers in the KeYmaera family. In: Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Ulbrich, M. (eds.) Deductive Software Verification: Future Perspectives. LNCS, vol. 12345, pp. 21–64. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64354-6_2
17. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reason. **41**(2), 143–189 (2008). https://doi.org/10.1007/s10817-008-9103-8
18. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14509-4
19. Platzer, A.: The complete proof theory of hybrid systems. In: LICS, pp. 541–550. IEEE Computer Society (2012). https://doi.org/10.1109/LICS.2012.64
20. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reason. **59**(2), 219–265 (2016). https://doi.org/10.1007/s10817-016-9385-1
21. Platzer, A.: Logical foundations of cyber-physical systems. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-63588-0
22. Platzer, A., Tan, Y.K.: Differential equation invariance axiomatization. J. ACM **67**(1) (2020). https://doi.org/10.1145/3380825
23. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation-based abstraction refinement. ACM Trans. Embed. Comput. Syst. **6**(1), 8 (2007). https://doi.org/10.1145/1210268.1210276
24. Richardson, D.: Some undecidable problems involving elementary functions of a real variable. J. Symb. Log. **33**(4), 514–520 (1968). https://doi.org/10.2307/2271358
25. Stengel, R.F.: Flight Dynamics. Princeton University Press (2004)
26. Tan, Y.K., Platzer, A.: An axiomatic approach to existence and liveness for differential equations. Form. Asp. Comput. **33**(4), 461–518 (2021). https://doi.org/10.1007/s00165-020-00525-0
27. Wiedijk, F.: Stateless HOL. In: Hirschowitz, T. (ed.) TYPES. EPTCS, vol. 53, pp. 47–61 (2009). https://doi.org/10.4204/EPTCS.53.4

Automatic Complexity Analysis of Integer Programs via Triangular Weakly Non-Linear Loops

Nils Lommen(✉), Fabian Meyer, and Jürgen Giesl(✉)

LuFG Informatik 2, RWTH Aachen University, Aachen, Germany
lommen@cs.rwth-aachen.de, giesl@informatik.rwth-aachen.de

Abstract. There exist several results on deciding termination and computing runtime bounds for *triangular weakly non-linear loops* (twn-loops). We show how to use results on such subclasses of programs where complexity bounds are computable within incomplete approaches for complexity analysis of full integer programs. To this end, we present a novel modular approach which computes local runtime bounds for subprograms which can be transformed into twn-loops. These local runtime bounds are then lifted to global runtime bounds for the whole program. The power of our approach is shown by our implementation in the tool KoAT which analyzes complexity of programs where all other state-of-the-art tools fail.

1 Introduction

Most approaches for automated complexity analysis of programs are based on incomplete techniques like ranking functions (see, e.g., [1–4,6,11,12,18,20,21,31]). However, there also exist numerous results on subclasses of programs where questions concerning termination or complexity are *decidable*, e.g., [5,14,15,19,22,24,25,32,34]. In this work we consider the subclass of *triangular weakly non-linear loops* (twn-loops), where there exist *complete* techniques for analyzing termination and runtime complexity (we discuss the "completeness" and decidability of these techniques below). An example for a twn-loop is:

$$\textbf{while } (x_1^2 + x_3^5 < x_2 \wedge x_1 \neq 0) \textbf{ do } (x_1, x_2, x_3) \leftarrow (-2 \cdot x_1,\ 3 \cdot x_2 - 2 \cdot x_3^3,\ x_3) \qquad (1)$$

Its guard is a propositional formula over (possibly *non-linear*) polynomial inequations. The update is *weakly non-linear*, i.e., no variable x_i occurs non-linear in its own update. Furthermore, it is *triangular*, i.e., we can order the variables such that the update of any x_i does not depend on the variables $x_1, \ldots, x_{i-1}$ with smaller indices. Then, by handling one variable after the other one can compute a *closed form* which corresponds to applying the loop's update n times. Using

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 235950644 (Project GI 274/6-2) and DFG Research Training Group 2236 UnRAVeL.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 734–754, 2022.
https://doi.org/10.1007/978-3-031-10769-6_43

these closed forms, termination can be reduced to an existential formula over $\mathbb{Z}$ [15] (whose validity is decidable for linear arithmetic and where SMT solvers often also prove (in)validity in the non-linear case). In this way, one can show that non-termination of twn-loops over $\mathbb{Z}$ is semi-decidable (and it is decidable over the real numbers).

While termination of twn-loops over $\mathbb{Z}$ is not decidable, by using the closed forms, [19] presented a "*complete*" complexity analysis technique. More precisely, for every twn-loop over $\mathbb{Z}$, it infers a polynomial which is an upper bound on the runtime for all those inputs where the loop terminates. So for all (possibly non-linear) terminating twn-loops over $\mathbb{Z}$, the technique of [19] *always* computes polynomial runtime bounds. In contrast, existing tools based on incomplete techniques for complexity analysis often fail for programs with non-linear arithmetic.

In [6,18] we presented such an incomplete modular technique for complexity analysis which uses individual ranking functions for different subprograms. Based on this, we now introduce a novel approach to automatically infer runtime bounds for programs possibly consisting of multiple consecutive or nested loops by handling some subprograms as twn-loops and by using ranking functions for others. In order to compute runtime bounds, we analyze subprograms in topological order, i.e., in case of multiple consecutive loops, we start with the first loop and propagate knowledge about the resulting values of variables to subsequent loops. By inferring runtime bounds for one subprogram after the other, in the end we obtain a bound on the runtime complexity of the whole program. We first try to compute runtime bounds for subprograms by so-called multiphase linear ranking functions (MΦRFs, see [3,4,18,20]). If MΦRFs do not yield a finite runtime bound for the respective subprogram, then we use our novel twn-technique on the unsolved parts of the subprogram. So for the first time, "complete" complexity analysis techniques like [19] for subclasses of programs with *non-linear* arithmetic are combined with incomplete techniques based on (linear) ranking functions like [6,18]. Based on our approach, in future work one could integrate "complete" techniques for further subclasses (e.g., for *solvable loops* [24,25,30,34] which can be transformed into twn-loops by suitable automorphisms [15]).

Structure: After introducing preliminaries in Sect. 2, in Sect. 3 we show how to lift a (local) runtime bound which is only sound for a subprogram to an overall global runtime bound. In contrast to previous techniques [6,18], our lifting approach works for any method of bound computation (not only for ranking functions). In Sect. 4, we improve the existing results on complexity analysis of twn- loops [14,15,19] such that they yield concrete polynomial bounds, we refine these bounds by considering invariants, and we show how to apply these results to full programs which contain twn-loops as subprograms. Section 5 extends this technique to larger subprograms which can be transformed into twn-loops. In Sect. 6 we evaluate the implementation of our approach in the complexity analysis tool KoAT and show that one can now also successfully analyze the runtime of programs containing non-linear arithmetic. We refer to [26] for all proofs.

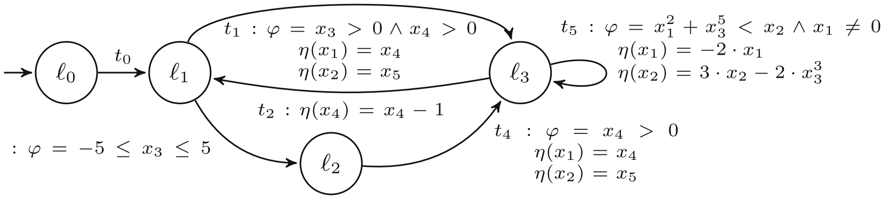

ℓ_0
t_0
ℓ_1
$t_1 : \varphi = x_3 > 0 \wedge x_4 > 0$
$\eta(x_1) = x_4$
$\eta(x_2) = x_5$
ℓ_3
$t_5 : \varphi = x_1^2 + x_3^5 < x_2 \wedge x_1 \neq 0$
$\eta(x_1) = -2 \cdot x_1$
$\eta(x_2) = 3 \cdot x_2 - 2 \cdot x_3^3$
$t_2 : \eta(x_4) = x_4 - 1$
$t_3 : \varphi = -5 \leq x_3 \leq 5$
ℓ_2
$t_4 : \varphi = x_4 > 0$
$\eta(x_1) = x_4$
$\eta(x_2) = x_5$

Fig. 1. An Integer Program with a Nested Self-Loop

2 Preliminaries

This section recapitulates preliminaries for complexity analysis from [6,18].

Definition 1 (Atoms and Formulas). *We fix a set $\mathcal{V}$ of variables. The set of* atoms $\mathcal{A}(\mathcal{V})$ *consists of all inequations $p_1 < p_2$ for polynomials $p_1, p_2 \in \mathbb{Z}[\mathcal{V}]$. $\mathcal{F}(\mathcal{V})$ is the set of all propositional* formulas *built from atoms $\mathcal{A}(\mathcal{V})$, $\wedge$, and $\vee$.*

In addition to "$<$", we also use "$\geq$", "$=$", "$\neq$", etc., and negations "$\neg$" which can be simulated by formulas (e.g., $p_1 \geq p_2$ is equivalent to $p_2 < p_1 + 1$ for integers).

For integer programs, we use a formalism based on transitions, which also allows us to represent **while**-programs like (1) easily. Our programs may have *non-deterministic branching*, i.e., the guards of several applicable transitions can be satisfied. Moreover, *non-deterministic sampling* is modeled by *temporary variables* whose values are updated arbitrarily in each evaluation step.

Definition 2 (Integer Program). $(\mathcal{PV}, \mathcal{L}, \ell_0, \mathcal{T})$ *is an* integer program *where*

- $\mathcal{PV} \subseteq \mathcal{V}$ *is a finite set of* program variables*, $\mathcal{V} \backslash \mathcal{PV}$ are* temporary variables
- $\mathcal{L}$ *is a finite set of* locations *with an* initial location $\ell_0 \in \mathcal{L}$
- $\mathcal{T}$ *is a finite set of* transitions*. A transition is a 4-tuple $(\ell, \varphi, \eta, \ell')$ with a* start location $\ell \in \mathcal{L}$*,* target location $\ell' \in \mathcal{L} \setminus \{\ell_0\}$*,* guard $\varphi \in \mathcal{F}(\mathcal{V})$*, and* update function $\eta : \mathcal{PV} \to \mathbb{Z}[\mathcal{V}]$ *mapping program variables to update polynomials.*

Transitions $(\ell_0, \_, \_, \_)$ are called *initial*. Note that ℓ_0 has no incoming transitions.

Example 3. Consider the program in Fig. 1 with $\mathcal{PV} = \{x_i \mid 1 \leq i \leq 5\}$, $\mathcal{L} = \{\ell_i \mid 0 \leq i \leq 3\}$, and $\mathcal{T} = \{t_i \mid 0 \leq i \leq 5\}$, where t_5 has non-linear arithmetic in its guard and update. We omitted trivial guards, i.e., $\varphi = \texttt{true}$, and identity updates of the form $\eta(v) = v$. Thus, t_5 corresponds to the **while**-program (1).

A *state* is a mapping $\sigma : \mathcal{V} \to \mathbb{Z}$, Σ denotes the set of all states, and $\mathcal{L} \times \Sigma$ is the set of *configurations*. We also apply states to arithmetic expressions p or formulas φ, where the number $\sigma(p)$ resp. the Boolean value $\sigma(\varphi)$ results from replacing each variable v by $\sigma(v)$. So for a state with $\sigma(x_1) = -8$, $\sigma(x_2) = 55$, and $\sigma(x_3) = 1$, the expression $x_1^2 + x_3^5$ evaluates to $\sigma(x_1^2 + x_3^5) = 65$ and the formula $\varphi = (x_1^2 + x_3^5 < x_2)$ evaluates to $\sigma(\varphi) = (65 < 55) = \texttt{false}$. From now on, we fix a program $(\mathcal{PV}, \mathcal{L}, \ell_0, \mathcal{T})$.

Definition 4 (Evaluation of Programs). *For configurations (ℓ, σ), (ℓ', σ') and $t = (\ell_t, \varphi, \eta, \ell'_t) \in \mathcal{T}$, $(\ell, \sigma) \to_t (\ell', \sigma')$ is an* evaluation *step if $\ell = \ell_t$, $\ell' = \ell'_t$, $\sigma(\varphi) = \texttt{true}$, and $\sigma(\eta(v)) = \sigma'(v)$ for all $v \in \mathcal{PV}$. Let $\to_{\mathcal{T}} = \bigcup_{t \in \mathcal{T}} \to_t$, where we also write $\to$ instead of $\to_t$ or $\to_{\mathcal{T}}$. Let $(\ell_0, \sigma_0) \to^k (\ell_k, \sigma_k)$ abbreviate $(\ell_0, \sigma_0) \to \ldots \to (\ell_k, \sigma_k)$ and let $(\ell, \sigma) \to^* (\ell', \sigma')$ if $(\ell, \sigma) \to^k (\ell', \sigma')$ for some $k \geq 0$.*

So when denoting states σ as tuples $(\sigma(x_1), \ldots, \sigma(x_5)) \in \mathbb{Z}^5$ for the program in Fig. 1, we have $(\ell_0, (1,5,7,1,3)) \to_{t_0} (\ell_1, (1,5,7,1,3)) \to_{t_1} (\ell_3, (1,1,3,1,3)) \to_{t_5}^3 (\ell_3, (1,-8,55,1,3)) \to_{t_2} \ldots$. The runtime complexity $\mathrm{rc}(\sigma_0)$ of a program corresponds to the length of the longest evaluation starting in the initial state σ_0.

Definition 5 (Runtime Complexity). *The* runtime complexity *is* $\mathrm{rc} : \Sigma \to \overline{\mathbb{N}}$ *with* $\overline{\mathbb{N}} = \mathbb{N} \cup \{\omega\}$ *and* $\mathrm{rc}(\sigma_0) = \sup\{k \in \mathbb{N} \mid \exists (\ell', \sigma').\, (\ell_0, \sigma_0) \to^k (\ell', \sigma')\}$.

3 Computing Global Runtime Bounds

We now introduce our general approach for computing (upper) runtime bounds. We use weakly monotonically increasing functions as bounds, since they can easily be "composed" (i.e., if f and g increase monotonically, then so does $f \circ g$).

Definition 6 (Bounds [6,18]). *The set of* bounds $\mathcal{B}$ *is the smallest set with $\overline{\mathbb{N}} \subseteq \mathcal{B}$, $\mathcal{PV} \subseteq \mathcal{B}$, and $\{b_1 + b_2,\ b_1 \cdot b_2,\ k^{b_1}\} \subseteq \mathcal{B}$ for all $k \in \mathbb{N}$ and $b_1, b_2 \in \mathcal{B}$.*

A bound constructed from $\mathbb{N}$, $\mathcal{PV}$, $+$, and $\cdot$ is *polynomial*. So for $\mathcal{PV} = \{x, y\}$, we have ω, x^2, $x + y$, $2^{x+y} \in \mathcal{B}$. Here, x^2 and $x + y$ are polynomial bounds.

We measure the size of variables by their absolute values. For any $\sigma \in \Sigma$, $|\sigma|$ is the state with $|\sigma|(v) = |\sigma(v)|$ for all $v \in \mathcal{V}$. So if σ_0 denotes the initial state, then $|\sigma_0|$ maps every variable to its initial "size", i.e., its initial absolute value. $\mathcal{RB}_{\mathrm{glo}} : \mathcal{T} \to \mathcal{B}$ is a *global runtime bound* if for each transition t and initial state $\sigma_0 \in \Sigma$, $\mathcal{RB}_{\mathrm{glo}}(t)$ evaluated in the state $|\sigma_0|$ over-approximates the number of evaluations of t in any run starting in the configuration (ℓ_0, σ_0). Let $\to_{\mathcal{T}}^* \circ \to_t$ denote the relation where arbitrary many evaluation steps are followed by a step with t.

Definition 7 (Global Runtime Bound [6,18]). *The function $\mathcal{RB}_{glo} : \mathcal{T} \to \mathcal{B}$ is a* global runtime bound *if for all $t \in \mathcal{T}$ and all states $\sigma_0 \in \Sigma$ we have $|\sigma_0|(\mathcal{RB}_{glo}(t)) \geq \sup\{k \in \mathbb{N} \mid \exists\, (\ell', \sigma').\, (\ell_0, \sigma_0)\, (\to_{\mathcal{T}}^* \circ \to_t)^k\, (\ell', \sigma')\}$.*

For the program in Fig. 1, in Example 12 we will infer $\mathcal{RB}_{\mathrm{glo}}(t_0) = 1$, $\mathcal{RB}_{\mathrm{glo}}(t_i) = x_4$ for $1 \leq i \leq 4$, and $\mathcal{RB}_{\mathrm{glo}}(t_5) = 8 \cdot x_4 \cdot x_5 + 13006 \cdot x_4$. By adding the bounds for all transitions, a global runtime bound $\mathcal{RB}_{\mathrm{glo}}$ yields an upper bound on the program's runtime complexity. So for all $\sigma_0 \in \Sigma$ we have $|\sigma_0|(\sum_{t \in \mathcal{T}} \mathcal{RB}_{\mathrm{glo}}(t)) \geq \mathrm{rc}(\sigma_0)$.

For *local runtime bounds*, we consider the *entry transitions* of subsets $\mathcal{T}' \subseteq \mathcal{T}$.

Definition 8 (Entry Transitions [6,18]**).** *Let* $\varnothing \neq \mathcal{T}' \subseteq \mathcal{T}$. *Its* entry transitions *are* $\mathcal{E}_{\mathcal{T}'} = \{t \mid t = (\ell, \varphi, \eta, \ell') \in \mathcal{T} \setminus \mathcal{T}' \wedge$ *there is a transition* $(\ell', \_, \_, \_) \in \mathcal{T}'\}$.

So in Fig. 1, we have $\mathcal{E}_{\mathcal{T} \setminus \{t_0\}} = \{t_0\}$ and $\mathcal{E}_{\{t_5\}} = \{t_1, t_4\}$.

In contrast to global runtime bounds, a *local* runtime bound $\mathcal{RB}_{\text{loc}} : \mathcal{E}_{\mathcal{T}'} \to \mathcal{B}$ only takes a subset $\mathcal{T}'$ into account. A *local run* is started by an entry transition $r \in \mathcal{E}_{\mathcal{T}'}$ followed by transitions from $\mathcal{T}'$. A *local runtime bound* considers a subset $\mathcal{T}'_{>} \subseteq \mathcal{T}'$ and over-approximates the number of evaluations of any transition from $\mathcal{T}'_{>}$ in an arbitrary local run of the subprogram with the transitions $\mathcal{T}'$. More precisely, for every $t \in \mathcal{T}'_{>}$, $\mathcal{RB}_{\text{loc}}(r)$ over-approximates the number of applications of t in any run of $\mathcal{T}'$, if $\mathcal{T}'$ is entered via $r \in \mathcal{E}_{\mathcal{T}'}$. However, local runtime bounds do not consider how often an entry transition from $\mathcal{E}_{\mathcal{T}'}$ is evaluated or how large a variable is when we evaluate an entry transition. To illustrate that $\mathcal{RB}_{\text{loc}}(r)$ is a bound on the number of evaluations of transitions from $\mathcal{T}'_{>}$ after evaluating r, we often write $\mathcal{RB}_{\text{loc}}(\to_r \mathcal{T}'_{>})$ instead of $\mathcal{RB}_{\text{loc}}(r)$.

Definition 9 (Local Runtime Bound). *Let* $\varnothing \neq \mathcal{T}'_{>} \subseteq \mathcal{T}' \subseteq \mathcal{T}$. *The function* $\mathcal{RB}_{loc} : \mathcal{E}_{\mathcal{T}'} \to \mathcal{B}$ *is a* local runtime bound *for* $\mathcal{T}'_{>}$ *w.r.t.* $\mathcal{T}'$ *if for all* $t \in \mathcal{T}'_{>}$, *all* $r \in \mathcal{E}_{\mathcal{T}'}$ *with* $r = (\ell, \_, \_, \_)$, *and all* $\sigma \in \Sigma$ *we have* $|\sigma|(\mathcal{RB}_{loc}(\to_r \mathcal{T}'_{>})) \geq \sup\{k \in \mathbb{N} \mid \exists\, \sigma_0, (\ell', \sigma').\ (\ell_0, \sigma_0) \to_{\mathcal{T}}^{*} \circ \to_r\ (\ell, \sigma)\ (\to_{\mathcal{T}'}^{*} \circ \to_t)^k\ (\ell', \sigma')\}$.

Our approach is *modular* since it computes local bounds for program parts separately. To lift local to global runtime bounds, we use *size bounds* $\mathcal{SB}(t, v)$ to over-approximate the size (i.e., absolute value) of the variable v after evaluating t in any run of the program. See [6] for the automatic computation of size bounds.

Definition 10 (Size Bound [6,18]**).** *The function* $\mathcal{SB} : (\mathcal{T} \times \mathcal{PV}) \to \mathcal{B}$ *is a* size bound *if for all* $(t, v) \in \mathcal{T} \times \mathcal{PV}$ *and all states* $\sigma_0 \in \Sigma$ *we have* $|\sigma_0|(\mathcal{SB}(t, v)) \geq \sup\{|\sigma'(v)| \mid \exists\, (\ell', \sigma').\ (\ell_0, \sigma_0)\ (\to^{*} \circ \to_t)\ (\ell', \sigma')\}$.

To compute global from local runtime bounds $\mathcal{RB}_{\text{loc}}(\to_r \mathcal{T}'_{>})$ and size bounds $\mathcal{SB}(r, v)$, Theorem 11 generalizes the approach of [6,18]. Each local run is started by an entry transition r. Hence, we use an already computed global runtime bound $\mathcal{RB}_{\text{glo}}(r)$ to over-approximate the number of times that such a local run is started. To over-approximate the size of each variable v when entering the local run, we instantiate it by the size bound $\mathcal{SB}(r, v)$. So size bounds on previous transitions are needed to compute runtime bounds, and similarly, runtime bounds are needed to compute size bounds in [6]. For any bound b, "$b\ [v/\mathcal{SB}(r, v) \mid v \in \mathcal{PV}]$" results from b by replacing every program variable v by $\mathcal{SB}(r, v)$. Here, weak monotonic increase of b ensures that the over-approximation of the variables v in b by $\mathcal{SB}(r, v)$ indeed also leads to an over-approximation of b. The analysis starts with an *initial* runtime bound $\mathcal{RB}_{\text{glo}}$ and an *initial* size bound $\mathcal{SB}$ which map all transitions resp. all pairs from $\mathcal{T} \times \mathcal{PV}$ to ω, except for the transitions t which do not occur in cycles of $\mathcal{T}$, where $\mathcal{RB}_{\text{glo}}(t) = 1$. Afterwards, $\mathcal{RB}_{\text{glo}}$ and $\mathcal{SB}$ are refined repeatedly, where we alternate between computing runtime and size bounds.

Theorem 11 (Computing Global Runtime Bounds). *Let $\mathcal{RB}_{glo}$ be a global runtime bound, $\mathcal{SB}$ be a size bound, and $\varnothing \neq \mathcal{T}'_{>} \subseteq \mathcal{T}' \subseteq \mathcal{T}$ such that $\mathcal{T}'$ contains no initial transitions. Moreover, let $\mathcal{RB}_{loc}$ be a local runtime bound for $\mathcal{T}'_{>}$ w.r.t. $\mathcal{T}'$. Then $\mathcal{RB}'_{glo}$ is also a global runtime bound, where for all $t \in \mathcal{T}$ we define:*

$$\mathcal{RB}'_{glo}(t) = \begin{cases} \mathcal{RB}_{glo}(t), & \text{if } t \in \mathcal{T} \setminus \mathcal{T}'_{>} \\ \sum_{r \in \mathcal{E}_{\mathcal{T}'}} \mathcal{RB}_{glo}(r) \cdot (\mathcal{RB}_{loc}(\to_r \mathcal{T}'_{>})\,[v/\mathcal{SB}(r,v) \mid v \in \mathcal{PV}]), & \text{if } t \in \mathcal{T}'_{>} \end{cases}$$

Example 12. For the example in Fig. 1, we first use $\mathcal{T}'_{>} = \{t_2\}$ and $\mathcal{T}' = \mathcal{T} \setminus \{t_0\}$. With the ranking function x_4 one obtains $\mathcal{RB}_{\mathrm{loc}}(\to_{t_0} \mathcal{T}'_{>}) = x_4$, since t_2 decreases the value of x_4 and no transition increases it. Then we can infer the global runtime bound $\mathcal{RB}_{\mathrm{glo}}(t_2) = \mathcal{RB}_{\mathrm{glo}}(t_0) \cdot (x_4\,[v/\mathcal{SB}(t_0, v) \mid v \in \mathcal{PV}]) = x_4$ as $\mathcal{RB}_{\mathrm{glo}}(t_0) = 1$ (since t_0 is evaluated at most once) and $\mathcal{SB}(t_0, x_4) = x_4$ (since t_0 does not change any variables). Similarly, we can infer $\mathcal{RB}_{\mathrm{glo}}(t_1) = \mathcal{RB}_{\mathrm{glo}}(t_3) = \mathcal{RB}_{\mathrm{glo}}(t_4) = x_4$.

For $\mathcal{T}'_{>} = \mathcal{T}' = \{t_5\}$, our twn-approach in Sect. 4 will infer the local runtime bound $\mathcal{RB}_{\mathrm{loc}} : \mathcal{E}_{\{t_5\}} \to \mathcal{B}$ with $\mathcal{RB}_{\mathrm{loc}}(\to_{t_1} \{t_5\}) = 4 \cdot x_2 + 3$ and $\mathcal{RB}_{\mathrm{loc}}(\to_{t_4} \{t_5\}) = 4 \cdot x_2 + 4 \cdot x_3^3 + 4 \cdot x_3^5 + 3$ in Example 30. By Theorem 11 we obtain the global bound

$$\begin{aligned} \mathcal{RB}_{\mathrm{glo}}(t_5) &= \mathcal{RB}_{\mathrm{glo}}(t_1) \cdot (\mathcal{RB}_{\mathrm{loc}}(\to_{t_1} \{t_5\})[v/\mathcal{SB}(t_1, v) \mid v \in \mathcal{PV}]) + \\ &\quad\ \mathcal{RB}_{\mathrm{glo}}(t_4) \cdot (\mathcal{RB}_{\mathrm{loc}}(\to_{t_4} \{t_5\})[v/\mathcal{SB}(t_4, v) \mid v \in \mathcal{PV}]) \\ &= x_4 \cdot (4 \cdot x_5 + 3) + x_4 \cdot (4 \cdot x_5 + 4 \cdot 5^3 + 4 \cdot 5^5 + 3) \\ &\qquad\qquad\qquad (\text{as } \mathcal{SB}(t_1, x_2) = \mathcal{SB}(t_4, x_2) = x_5 \text{ and } \mathcal{SB}(t_4, x_3) = 5) \\ &= 8 \cdot x_4 \cdot x_5 + 13006 \cdot x_4. \end{aligned}$$

Thus, $\mathrm{rc}(\sigma_0) \in \mathcal{O}(n^2)$ where n is the largest initial absolute value of all program variables. While the approach of [6,18] was limited to local bounds resulting from ranking functions, here we need our Theorem 11. It allows us to use both local bounds resulting from twn-loops (for the non-linear transition t_5 where tools based on ranking functions cannot infer a bound, see Sect. 6) and local bounds resulting from ranking functions (for $t_1, \ldots, t_4$, since our twn-approach of Sect. 4 and 5 is limited to so-called simple cycles and cannot handle the full program).

In contrast to [6,18], we allow different local bounds for different entry transitions in Definition 9 and Theorem 11. Our example demonstrates that this can indeed lead to a smaller asymptotic bound for the whole program: By distinguishing the cases where t_5 is reached via t_1 or t_4, we end up with a quadratic bound, because the local bound $\mathcal{RB}_{\mathrm{loc}}(\to_{t_1} \{t_5\})$ is linear and while x_3 occurs with degrees 5 and 3 in $\mathcal{RB}_{\mathrm{loc}}(\to_{t_4} \{t_5\})$, the size bound for x_3 is constant after t_3 and t_4.

To improve size and runtime bounds repeatedly, we treat the strongly connected components (SCCs)[1] of the program in topological order such that

[1] As usual, a graph is *strongly connected* if there is a path from every node to every other node. A *strongly connected component* is a maximal strongly connected subgraph.

improved bounds for previous transitions are already available when handling the next SCC. We first try to infer local runtime bounds by multiphase-linear ranking functions (see [18] which also contains a heuristic for choosing $\mathcal{T}'_>$ and $\mathcal{T}'$ when using ranking functions). If ranking functions do not yield finite local bounds for all transitions of the SCC, then we apply the twn-technique from Sect. 4 and 5 on the remaining unbounded transitions (see Sect. 5 for choosing $\mathcal{T}'_>$ and $\mathcal{T}'$ in that case). Afterwards, the global runtime bound is updated according to Theorem 11.

4 Local Runtime Bounds for Twn-Self-Loops

In Sect. 4.1 we recapitulate twn-loops and their termination in our setting. Then in Sect. 4.2 we present a (complete) algorithm to infer polynomial runtime bounds for all terminating twn-loops. Compared to [19], we increased its precision considerably by computing bounds that take the different roles of the variables into account and by using over-approximations to remove monomials. Moreover, we show how our algorithm can be used to infer local runtime bounds for twn-loops occurring in integer programs. Section 5 will show that our algorithm can also be applied to infer runtime bounds for larger cycles in programs instead of just self-loops.

4.1 Termination of Twn-Loops

Definition 13 extends the definition of twn-loops in [15,19] by an initial transition and an update-invariant. Here, ψ is an *update-invariant* if $\models \psi \to \eta(\psi)$ where η is the update of the transition (i.e., invariance must hold independent of the guard).

Definition 13. (Twn-Loop). *An integer program $(\mathcal{PV}, \mathcal{L}, \ell_0, \mathcal{T})$ is a* triangular weakly non-linear loop (twn-loop) *if $\mathcal{PV} = \{x_1, \ldots, x_d\}$ for some $d \geq 1$, $\mathcal{L} = \{\ell_0, \ell\}$, and $\mathcal{T} = \{t_0, t\}$ with $t_0 = (\ell_0, \psi, \mathrm{id}, \ell)$ and $t = (\ell, \varphi, \eta, \ell)$ for some $\psi, \varphi \in \mathcal{F}(\mathcal{PV})$ with $\models \psi \to \eta(\psi)$, where $\mathrm{id}(v) = v$ for all $v \in \mathcal{PV}$, and for all $1 \leq i \leq d$ we have $\eta(x_i) = c_i \cdot x_i + p_i$ for some $c_i \in \mathbb{Z}$ and some polynomial $p_i \in \mathbb{Z}[x_{i+1}, \ldots, x_d]$. We often denote the loop by (ψ, φ, η) and refer to ψ, φ, η as its (update-) invariant, guard, and update, respectively. If $c_i \geq 0$ holds for all $1 \leq i \leq d$, then the program is a* non-negative *triangular weakly non-linear loop* (tnn-loop)*.*

Example 14. The program consisting of the initial transition $(\ell_0, \texttt{true}, \mathrm{id}, \ell_3)$ and the self-loop t_5 in Fig. 1 is a twn-loop (corresponding to the **while**-loop (1)). This loop terminates as every iteration increases x_1^2 by a factor of 4 whereas x_2 is only tripled. Thus, $x_1^2 + x_3^5$ eventually outgrows the value of x_2.

To transform programs into twn- or tnn-form, one can combine subsequent transitions by *chaining*. Here, similar to states σ, we also apply the update η to polynomials and formulas by replacing each program variable v by $\eta(v)$.

Definition 15 (Chaining). *Let $t_1, \dots, t_n$ be a sequence of transitions without temporary variables where $t_i = (\ell_i, \varphi_i, \eta_i, \ell_{i+1})$ for all $1 \leq i \leq n-1$, i.e., the target location of t_i is the start location of t_{i+1}. We may have $t_i = t_j$ for $i \neq j$, i.e., a transition may occur several times in the sequence. Then the transition $t_1 \star \dots \star t_n = (\ell_1, \varphi, \eta, \ell_{n+1})$ results from* chaining *$t_1, \dots, t_n$ where*

$$\varphi = \varphi_1 \,\wedge\, \eta_1(\varphi_2) \,\wedge\, \eta_2(\eta_1(\varphi_3)) \,\wedge\, \dots \,\wedge\, \eta_{n-1}(\dots \eta_1(\varphi_n) \dots)$$
$$\eta(v) = \eta_n(\dots \eta_1(v) \dots) \text{ for all } v \in \mathcal{PV}, \text{ i.e., } \eta = \eta_n \circ \dots \circ \eta_1.$$

Similar to [15,19], we can restrict ourselves to tnn-loops, since chaining transforms any twn-loop L into a tnn-loop $L \star L$. Chaining preserves the termination behavior, and a bound on $L \star L$'s runtime can be transformed into a bound for L.

Lemma 16 (Chaining Preserves Asymptotic Runtime, see [19, Lemma 18]). *For the twn-loop $L = (\psi, \varphi, \eta)$ with the transitions $t_0 = (\ell_0, \psi, \mathrm{id}, \ell)$, $t = (\ell, \varphi, \eta, \ell)$, and runtime complexity rc_L, the program $L \star L$ with the transitions t_0 and $t \star t = (\psi, \varphi \wedge \eta(\varphi), \eta \circ \eta)$ is a tnn-loop. For its runtime complexity $\mathrm{rc}_{L\star L}$, we have $2 \cdot \mathrm{rc}_{L\star L}(\sigma) \;\leq\; \mathrm{rc}_L(\sigma) \;\leq\; 2 \cdot \mathrm{rc}_{L\star L}(\sigma) + 1$ for all $\sigma \in \Sigma$.*

Example 17. The program of Example 14 is only a twn-loop and not a tnn-loop as x_1 occurs with a negative coefficient -2 in its own update. Hence, we chain the loop and consider $t_5 \star t_5$. The update of $t_5 \star t_5$ is $(\eta \circ \eta)(x_1) = 4 \cdot x_1$, $(\eta \circ \eta)(x_2) = 9 \cdot x_2 - 8 \cdot x_3^3$, and $(\eta \circ \eta)(x_3) = x_3$. To ease the presentation, in this example we will keep the guard φ instead of using $\varphi \wedge \eta(\varphi)$ (ignoring $\eta(\varphi)$ in the conjunction of the guard does not decrease the runtime complexity).

Our algorithm starts with computing a closed form for the loop update, which describes the values of the program variables after n iterations of the loop. Formally, a tuple of arithmetic expressions $\mathtt{cl}_{\boldsymbol{x}}^n = (\mathtt{cl}_{x_1}^n, \dots, \mathtt{cl}_{x_d}^n)$ over the variables $\boldsymbol{x} = (x_1, \dots, x_d)$ and the distinguished variable n is a *(normalized) closed form* for the update η with *start value* $n_0 \geq 0$ if for all $1 \leq i \leq d$ and all $\sigma : \{x_1, \dots, x_d, n\} \to \mathbb{Z}$ with $\sigma(n) \geq n_0$, we have $\sigma(\mathtt{cl}_{x_i}^n) = \sigma(\eta^n(x_i))$. As shown in [14,15,19], for tnn-loops such a normalized closed form and the start value n_0 can be computed by handling one variable after the other, and these normalized closed forms can be represented as so-called *normalized poly-exponential expressions*. Here, $\mathbb{N}_{\geq m}$ stands for $\{x \in \mathbb{N} \mid x \geq m\}$.

Definition 18. (Normalized Poly-Exponential Expression [14,15,19]). *Let $\mathcal{PV} = \{x_1, \dots, x_d\}$. Then we define the set of all* normalized poly-exponential expressions *by $\mathbb{NPE} = \{\sum_{j=1}^{\ell} p_j \cdot n^{a_j} \cdot b_j^n \;\big|\; \ell, a_j \in \mathbb{N},\ p_j \in \mathbb{Q}[\mathcal{PV}],\ b_j \in \mathbb{N}_{\geq 1}\}$.*

Example 19. A normalized closed form (with start value $n_0 = 0$) for the tnn-loop in Example 17 is $\mathtt{cl}_{x_1}^n = x_1 \cdot 4^n$, $\mathtt{cl}_{x_2}^n = (x_2 - x_3^3) \cdot 9^n + x_3^3$, and $\mathtt{cl}_{x_3}^n = x_3$.

Using the normalized closed form, similar to [15] one can represent non-termination of a tnn-loop (ψ, φ, η) by the formula

$$\exists\, \boldsymbol{x} \in \mathbb{Z}^d,\ m \in \mathbb{N}.\ \forall n \in \mathbb{N}_{\geq m}.\ \psi \wedge \varphi[\boldsymbol{x}/\mathtt{cl}_{\boldsymbol{x}}^n]. \tag{2}$$

Here, $\varphi[\boldsymbol{x}/\texttt{cl}^n_{\boldsymbol{x}}]$ means that each variable x_i in φ is replaced by $\texttt{cl}^n_{x_i}$. Since ψ is an update-invariant, if ψ holds, then $\psi[\boldsymbol{x}/\texttt{cl}^n_{\boldsymbol{x}}]$ holds as well for all $n \geq n_0$. Hence, whenever $\forall n \in \mathbb{N}_{\geq m}.\ \psi \wedge \varphi[\boldsymbol{x}/\texttt{cl}^n_{\boldsymbol{x}}]$ holds, then $\texttt{cl}_{\boldsymbol{x}}^{\max\{n_0,m\}}$ witnesses non-termination. Thus, invalidity of (2) is equivalent to termination of the loop.

Normalized poly-exponential expressions have the advantage that it is always clear which addend determines their asymptotic growth when increasing n. So as in [15], (2) can be transformed into an existential formula and we use an SMT solver to prove its invalidity in order to prove termination of the loop. As shown in [15, Theorem 42], non-termination of twn-loops over $\mathbb{Z}$ is semi-decidable and deciding termination is Co-NP-complete if the loop is linear and the eigenvalues of the update matrix are rational.

4.2 Runtime Bounds for Twn-Loops via Stabilization Thresholds

As observed in [19], since the closed forms for tnn-loops are poly-exponential expressions that are weakly monotonic in n, every tnn-loop (ψ, φ, η) *stabilizes* for each input $\boldsymbol{e} \in \mathbb{Z}^d$. So there is a number of loop iterations (a *stabilization threshold* $\mathrm{sth}_{(\psi,\varphi,\eta)}(\boldsymbol{e})$), such that the truth value of the loop guard φ does not change anymore when performing further loop iterations. Hence, the runtime of every terminating tnn-loop is bounded by its stabilization threshold.

Definition 20 (Stabilization Threshold). *Let (ψ, φ, η) be a tnn-loop with $\mathcal{PV} = \{x_1, \ldots, x_d\}$. For each $\boldsymbol{e} = (e_1, \ldots, e_d) \in \mathbb{Z}^d$, let $\sigma_{\boldsymbol{e}} \in \varSigma$ with $\sigma_{\boldsymbol{e}}(x_i) = e_i$ for all $1 \leq i \leq d$. Let $\varPsi \subseteq \mathbb{Z}^d$ such that $\boldsymbol{e} \in \varPsi$ iff $\sigma_{\boldsymbol{e}}(\psi)$ holds. Then $\mathrm{sth}_{(\psi,\varphi,\eta)} : \mathbb{Z}^d \to \mathbb{N}$ is the* stabilization threshold *of (ψ, φ, η) if for all $\boldsymbol{e} \in \varPsi$, $\mathrm{sth}_{(\psi,\varphi,\eta)}(\boldsymbol{e})$ is the smallest number such that $\sigma_{\boldsymbol{e}}\left(\, \eta^n(\varphi) \leftrightarrow \eta^{\mathrm{sth}_{(\psi,\varphi,\eta)}(\boldsymbol{e})}(\varphi)\,\right)$ holds for all $n \geq \mathrm{sth}_{(\psi,\varphi,\eta)}(\boldsymbol{e})$.*

For the tnn-loop from Example 17, it will turn out that $2 \cdot x_2 + 2 \cdot x_3^3 + 2 \cdot x_3^5 + 1$ is an upper bound on its stabilization threshold, see Example 28.

To compute such upper bounds on a tnn-loop's stabilization threshold (i.e., upper bounds on its runtime if the loop is terminating), we now present a construction based on *monotonicity thresholds*, which are computable [19, Lemma 12].

Definition 21 (Monotonicity Threshold [19]). *Let $(b_1, a_1), (b_2, a_2) \in \mathbb{N}^2$ such that $(b_1, a_1) >_{\text{lex}} (b_2, a_2)$ (i.e., $b_1 > b_2$ or both $b_1 = b_2$ and $a_1 > a_2$). For any $k \in \mathbb{N}_{\geq 1}$, the k-*monotonicity threshold *of (b_1, a_1) and (b_2, a_2) is the smallest $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ we have $n^{a_1} \cdot b_1^n > k \cdot n^{a_2} \cdot b_2^n$.*

For example, the 1-monotonicity threshold of $(4, 0)$ and $(3, 1)$ is 7 as the largest root of $f(n) = 4^n - n \cdot 3^n$ is approximately 6.5139.

Our procedure again instantiates the variables of the loop guard φ by the normalized closed form $\texttt{cl}^n_{\boldsymbol{x}}$ of the loop's update. However, in the poly-exponential expressions $\sum_{j=1}^{\ell} p_j \cdot n^{a_j} \cdot b_j^n$ resulting from $\varphi[\boldsymbol{x}/\texttt{cl}^n_{\boldsymbol{x}}]$, the corresponding technique of [19, Lemma 21] over-approximated the polynomials p_j by a polynomial that did not distinguish the effects of the different variables $x_1, \dots, x_d$. Such an over-approximation is only useful for a direct asymptotic bound on the runtime of the twn-loop, but it is too coarse for a useful *local* runtime bound within the complexity analysis of a larger program. For instance, in Example 12 it is crucial to obtain local bounds like $4 \cdot x_2 + 4 \cdot x_3^3 + 4 \cdot x_3^5 + 3$ which indicate that only the variable x_3 may influence the runtime with an exponent of 3 or 5. Thus, if the size of x_3 is bound by a constant, then the resulting global bound becomes linear.

So we now improve precision and over-approximate the polynomials p_j by the polynomial $\sqcup\{p_1, \dots, p_\ell\}$ which contains every monomial $x_1^{e_1} \cdot \ldots \cdot x_d^{e_d}$ of $\{p_1, \dots, p_\ell\}$, using the absolute value of the largest coefficient with which the monomial occurs in $\{p_1, \dots, p_\ell\}$. Thus, $\sqcup\{x_3^3 - x_3^5, x_2 - x_3^3\} = x_2 + x_3^3 + x_3^5$. In the following let $\boldsymbol{x} = (x_1, \dots, x_d)$, and for $\boldsymbol{e} = (e_1, \dots, e_d) \in \mathbb{N}^d$, $\boldsymbol{x}^{\boldsymbol{e}}$ denotes $x_1^{e_1} \cdot \ldots \cdot x_d^{e_d}$.

Definition 22 (Over-Approximation of Polynomials). *Let $p_1, \dots, p_\ell \in \mathbb{Z}[\boldsymbol{x}]$, and for all $1 \leq j \leq \ell$, let $\mathcal{I}_j \subseteq (\mathbb{Z} \setminus \{0\}) \times \mathbb{N}^d$ be the* index set *of the polynomial p_j where $p_j = \sum_{(c,\boldsymbol{e}) \in \mathcal{I}_j} c \cdot \boldsymbol{x}^{\boldsymbol{e}}$ and there are no $c \neq c'$ with $(c, \boldsymbol{e}), (c', \boldsymbol{e}) \in \mathcal{I}_j$. For all $\boldsymbol{e} \in \mathbb{N}^d$ we define $c_{\boldsymbol{e}} \in \mathbb{N}$ with $c_{\boldsymbol{e}} = \max\{|c| \mid (c, \boldsymbol{e}) \in \mathcal{I}_1 \cup \ldots \cup \mathcal{I}_\ell\}$, where $\max \varnothing = 0$. Then the* over-approximation *of $p_1, \dots, p_\ell$ is $\sqcup\{p_1, \dots, p_\ell\} = \sum_{\boldsymbol{e} \in \mathbb{N}^d} c_{\boldsymbol{e}} \cdot \boldsymbol{x}^{\boldsymbol{e}}$.*

Clearly, $\sqcup\{p_1, \dots, p_\ell\}$ indeed over-approximates the absolute value of each p_j.

Corollary 23 (Soundness of $\sqcup\{p_1, \dots, p_\ell\}$). *For all $\sigma : \{x_1, \dots, x_d\} \to \mathbb{Z}$ and all $1 \leq j \leq \ell$, we have $|\sigma|(\sqcup\{p_1, \dots, p_\ell\}) \geq |\sigma(p_j)|$.*

A drawback is that $\sqcup\{p_1, \dots, p_\ell\}$ considers all monomials and to obtain weakly monotonically increasing bounds from $\mathcal{B}$, it uses the absolute values of their coefficients. This can lead to polynomials of unnecessarily high degree. To improve the precision of the resulting bounds, we now allow to over-approximate the poly-exponential expressions $\sum_{j=1}^{\ell} p_j \cdot n^{a_j} \cdot b_j^n$ which result from instantiating the variables of the loop guard by the closed form. For this over-approximation, we take the invariant ψ of the tnn-loop into account. So while (2) showed that update-invariants ψ can restrict the sets of possible witnesses for non-termination and thus simplify the termination proofs of twn-loops, we now show that preconditions ψ can also be useful to improve the bounds on twn-loops.

More precisely, Definition 24 allows us to replace addends $p \cdot n^a \cdot b^n$ by $p \cdot n^i \cdot j^n$ where $(j, i) >_{\text{lex}} (b, a)$ if the monomial p is always positive (when the precondition ψ is fulfilled) and where $(b, a) >_{\text{lex}} (i, j)$ if p is always non-positive.

Definition 24 (Over-Approximation of Poly-Exponential Expressions). *Let $\psi \in \mathcal{F}(\mathcal{PV})$ and let $npe = \sum_{(p,a,b) \in \Lambda} p \cdot n^a \cdot b^n \in \mathbb{NPE}$ where Λ is a set of tuples (p, a, b) containing a monomial*[2] *p and two numbers $a, b \in \mathbb{N}$. Here, we*

[2] Here, we consider monomials of the form $p = c \cdot x_1^{e_1} \cdot \ldots \cdot x_d^{e_d}$ with coefficients $c \in \mathbb{Q}$.

may have $(p,a,b),(p',a,b) \in \Lambda$ *for* $p \neq p'$. *Let* $\Delta, \Gamma \subseteq \Lambda$ *such that* $\models \psi \to (p > 0)$ *holds for all* $(p,a,b) \in \Delta$ *and* $\models \psi \to (p \leq 0)$ *holds for all* $(p,a,b) \in \Gamma$.[3] *Then*

$$\lceil npe \rceil^{\psi}_{\Delta,\Gamma} = \sum\nolimits_{(p,a,b)\in\Delta\uplus\Gamma} p \cdot n^{i_{(p,a,b)}} \cdot j^{n}_{(p,a,b)} + \sum\nolimits_{(p,a,b)\in\Lambda\setminus(\Delta\uplus\Gamma)} p \cdot n^{a} \cdot b^{n}$$

is an over-approximation *of npe if* $i_{(p,a,b)}, j_{(p,a,b)} \in \mathbb{N}$ *are numbers such that* $(j_{(p,a,b)}, i_{(p,a,b)}) >_{\text{lex}} (b,a)$ *holds if* $(p,a,b) \in \Delta$ *and* $(b,a) >_{\text{lex}} (j_{(p,a,b)}, i_{(p,a,b)})$ *holds if* $(p,a,b) \in \Gamma$. *Note that* $i_{(p,a,b)}$ *or* $j_{(p,a,b)}$ *can also be* 0.

Example 25. Let $npe = q_3 \cdot 16^n + q_2 \cdot 9^n + q_1 = q_3 \cdot 16^n + q_2' \cdot 9^n + q_2'' \cdot 9^n + q_1' + q_1''$, where $q_3 = -x_1^2$, $q_2 = q_2' + q_2''$, $q_2' = x_2$, $q_2'' = -x_3^3$, $q_1 = q_1' + q_1''$, $q_1' = x_3^3$, $q_1'' = -x_3^5$, and $\psi = (x_3 > 0)$. We can choose $\Delta = \{(x_3^3, 0, 1)\}$ since $\models \psi \to (x_3^3 > 0)$ and $\Gamma = \{(-x_3^5, 0, 1)\}$ since $\models \psi \to (-x_3^5 \leq 0)$. Moreover, we choose $j_{(x_3^3,0,1)} = 9$, $i_{(x_3^3,0,1)} = 0$, which is possible since $(9,0) >_{\text{lex}} (1,0)$. Similarly, we choose $j_{(-x_3^5,0,1)} = 0$, $i_{(-x_3^5,0,1)} = 0$, since $(1,0) >_{\text{lex}} (0,0)$. Thus, we replace x_3^3 and $-x_3^5$ by the larger addends $x_3^3 \cdot 9^n$ and 0. The motivation for the latter is that this removes all addends with exponent 5 from *npe*. The motivation for the former is that then, we have both the addends $-x_3^3 \cdot 9^n$ and $x_3^3 \cdot 9^n$ in the expression which cancel out, i.e., this removes all addends with exponent 3. Hence, we obtain $\lceil npe \rceil^{\psi}_{\Delta,\Gamma} = p_2 \cdot 16^n + p_1 \cdot 9^n$ with $p_2 = -x_1^2$ and $p_1 = x_2$. To find a suitable over-approximation which removes addends with high exponents, our implementation uses a heuristic for the choice of Δ, Γ, $i_{(p,a,b)}$, and $j_{(p,a,b)}$.

The following lemma shows the soundness of the over-approximation $\lceil npe \rceil^{\psi}_{\Delta,\Gamma}$.

Lemma 26 (Soundness of $\lceil npe \rceil^{\psi}_{\Delta,\Gamma}$). *Let* ψ, *npe*, Δ, Γ, $i_{(p,a,b)}$, $j_{(p,a,b)}$, *and* $\lceil npe \rceil^{\psi}_{\Delta,\Gamma}$ *be as in Definition 24, and let* $D_{\lceil npe \rceil^{\psi}_{\Delta,\Gamma}} =$

$$\begin{aligned} \max(\ &\{1\text{-}\textit{monotonicity threshold of } (j_{(p,a,b)}, i_{(p,a,b)}) \textit{ and } (b,a) \mid (p,a,b) \in \Delta\} \\ &\cup \{1\text{-}\textit{monotonicity threshold of } (b,a) \textit{ and } (j_{(p,a,b)}, i_{(p,a,b)}) \mid (p,a,b) \in \Gamma\}). \end{aligned}$$

Then for all $\boldsymbol{e} \in \Psi$ *and all* $n \geq D_{\lceil npe \rceil^{\psi}_{\Delta,\Gamma}}$, *we have* $\sigma_{\boldsymbol{e}}(\lceil npe \rceil^{\psi}_{\Delta,\Gamma}) \geq \sigma_{\boldsymbol{e}}(npe)$.

For any terminating tnn-loop (ψ, φ, η), Theorem 27 now uses the new concepts of Definition 22 and 24 to compute a polynomial $\text{sth}^{\sqcup}$ which is an upper bound on the loop's stabilization threshold (and hence, on its runtime). For any atom $\alpha = (s_1 < s_2)$ (resp. $s_2 - s_1 > 0$) in the loop guard φ, let $npe_\alpha \in \mathbb{NPE}$ be a poly-exponential expression which results from multiplying $(s_2 - s_1)[\boldsymbol{x}/\texttt{cl}^n_{\boldsymbol{x}}]$ with the least common multiple of all denominators occurring in $(s_2 - s_1)[\boldsymbol{x}/\texttt{cl}^n_{\boldsymbol{x}}]$. Since the loop is terminating, for some of these atoms this expression will become non-positive for large enough n and our goal is to compute bounds on their corresponding stabilization thresholds. First, one can replace npe_α by an over-approximation $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}$ where $\psi' = (\psi \wedge \varphi)$ considers both the invariant ψ

[3] Δ and Γ do not have to contain *all* such tuples, but can be (possibly empty) subsets.

and the guard φ. Let $\Psi' \subseteq \mathbb{Z}^d$ such that $\boldsymbol{e} \in \Psi'$ iff $\sigma_{\boldsymbol{e}}(\psi')$ holds. By Lemma 26 (i.e., $\sigma_{\boldsymbol{e}}(\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}) \geq \sigma_{\boldsymbol{e}}(npe_\alpha)$ for all $\boldsymbol{e} \in \Psi'$), it suffices to compute a bound on the stabilization threshold of $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}$ if it is always non-positive for large enough n, because if $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}$ is non-positive, then so is npe_α. We say that an over-approximation $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}$ is *eventually non-positive* iff whenever $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma} \neq npe_\alpha$, then one can show that for all $\boldsymbol{e} \in \Psi'$, $\sigma_{\boldsymbol{e}}(\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma})$ is always non-positive for large enough n.[4] Using over-approximations $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}$ can be advantageous because $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}$ may contain less monomials than npe_α and thus, the construction $\sqcup$ from Definition 22 can yield a polynomial of lower degree. So although npe_α's stabilization threshold might be smaller than the one of $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}$, our technique might compute a smaller bound on the stabilization threshold when considering $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}$ instead of npe.

Theorem 27 (Bound on Stabilization Threshold). *Let $L = (\psi, \varphi, \eta)$ be a terminating tnn-loop, let $\psi' = (\psi \wedge \varphi)$, and let $\texttt{cl}^n_{\boldsymbol{x}}$ be a normalized closed form for η with start value n_0. For every atom $\alpha = (s_1 < s_2)$ in φ, let $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}$ be an eventually non-positive over-approximation of npe_α and let $D_\alpha = D_{\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma}}$.*

If $\lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma} = \sum_{j=1}^{\ell} p_j \cdot n^{a_j} \cdot b_j^n$ with $p_j \neq 0$ for all $1 \leq j \leq \ell$ and $(b_\ell, a_\ell) >_{\text{lex}} \ldots >_{\text{lex}} (b_1, a_1)$, then let $C_\alpha = \max\{1, N_2, M_2, \ldots, N_\ell, M_\ell\}$, where we have:

$$M_j = \begin{cases} 0, & \textit{if } b_j = b_{j-1} \\ \textit{1-monotonicity threshold of} & \\ \quad (b_j, a_j) \textit{ and } (b_{j-1}, a_{j-1}+1), & \textit{if } b_j > b_{j-1} \end{cases} \qquad N_j = \begin{cases} 1, & \textit{if } j = 2 \\ mt', & \textit{if } j = 3 \\ \max\{mt, mt'\}, & \textit{if } j > 3 \end{cases}$$

Here, mt' is the $(j-2)$-monotonicity threshold of (b_{j-1}, a_{j-1}) and (b_{j-2}, a_{j-2}) and $mt = \max\{1\text{-monotonicity threshold of } (b_{j-2}, a_{j-2}) \text{ and } (b_i, a_i) \mid 1 \leq i \leq j-3\}$. Let $Pol_\alpha = \{p_1, \ldots, p_{\ell-1}\}$, $Pol = \bigcup_{atom\ \alpha\ occurs\ in\ \varphi} Pol_\alpha$, $C = \max\{C_\alpha \mid$ atom α occurs in $\varphi\}$, $D = \max\{D_\alpha \mid$ atom α occurs in $\varphi\}$, and $\text{sth}^{\sqcup} \in \mathbb{Z}[\boldsymbol{x}]$ with $\text{sth}^{\sqcup} = 2 \cdot \sqcup Pol + \max\{n_0, C, D\}$. Then for all $\boldsymbol{e} \in \Psi'$, we have $|\sigma_{\boldsymbol{e}}|(\text{sth}^{\sqcup}) \geq \text{sth}_{(\psi,\varphi,\eta)}(\boldsymbol{e})$. If the tnn-loop has the initial transition t_0 and looping transition t, then $\mathcal{RB}_{glo}(t_0) = 1$ and $\mathcal{RB}_{glo}(t) = \text{sth}^{\sqcup}$ is a global runtime bound for L.

Example 28. The guard φ of the tnn-loop in Example 17 has the atoms $\alpha = (x_1^2 + x_3^5 < x_2)$, $\alpha' = (0 < x_1)$, and $\alpha'' = (0 < -x_1)$ (since $x_1 \neq 0$ is transformed into $\alpha' \vee \alpha''$). When instantiating the variables by the closed forms of Example 19 with start value $n_0 = 0$, Theorem 27 computes the bound 1 on the stabilization thresholds for α' and α''. So the only interesting atom is $\alpha = (0 < s_2 - s_1)$ for $s_1 = x_1^2 + x_3^5$ and $s_2 = x_2$. We get $npe_\alpha = (s_2 - s_1)[\boldsymbol{x}/\texttt{cl}^n_{\boldsymbol{x}}] = q_3 \cdot 16^n + q_2 \cdot 9^n + q_1$, with q_j as in Example 25.

[4] This can be shown similar to the proof of (2) for (non-)termination of the loop. Thus, we transform $\exists\, \boldsymbol{x} \in \mathbb{Z}^d,\ m \in \mathbb{N}.\ \forall n \in \mathbb{N}_{\geq m}.\ \psi' \wedge \lceil npe_\alpha \rceil^{\psi'}_{\Delta,\Gamma} > 0$ into an existential formula as in [15] and try to prove its invalidity by an SMT solver.

In the program of Fig. 1, the corresponding self-loop t_5 has two entry transitions t_4 and t_1 which result in two tnn-loops with the update-invariants $\psi_1 = \texttt{true}$ resulting from transition t_4 and $\psi_2 = (x_3 > 0)$ from t_1. So ψ_2 is an update-invariant of t_5 which always holds when reaching t_5 via transition t_1.

For $\psi_1 = \texttt{true}$, we choose $\Delta = \Gamma = \varnothing$, i.e., $\lceil npe_\alpha \rceil_{\Delta,\Gamma}^{\psi_1'} = npe_\alpha$. So we have $b_3 = 16$, $b_2 = 9$, $b_1 = 1$, and $a_j = 0$ for all $1 \leq j \leq 3$. We obtain

$$\begin{aligned} M_2 &= 0, \text{ as 0 is the 1-monotonicity threshold of } (9,0) \text{ and } (1,1)\\ M_3 &= 0, \text{ as 0 is the 1-monotonicity threshold of } (16,0) \text{ and } (9,1)\\ N_2 = 1 \;\text{ and }\; N_3 &= 1, \text{ as 1 is the 1-monotonicity threshold of } (9,0) \text{ and } (1,0). \end{aligned}$$

Hence, we get $C = C_\alpha = \max\{1, N_2, M_2, N_3, M_3\} = 1$. So we obtain the runtime bound $\mathrm{sth}_{\psi_1}^{\sqcup} = 2 \cdot \sqcup\{q_1, q_2\} + \max\{n_0, C_\alpha\} = 2 \cdot x_2 + 2 \cdot x_3^3 + 2 \cdot x_3^5 + 1$ for the loop $t_5 \star t_5$ w.r.t. ψ_1. By Lemma 16, this means that $2 \cdot \mathrm{sth}_{\psi_1}^{\sqcup} + 1 = 4 \cdot x_2 + 4 \cdot x_3^3 + 4 \cdot x_3^5 + 3$ is a runtime bound for the loop at transition t_5.

For the update-invariant $\psi_2 = (x_3 > 0)$, we use the over-approximation $\lceil npe_\alpha \rceil_{\Delta,\Gamma}^{\psi_2'} = p_2 \cdot 16^n + p_1 \cdot 9^n$ with $p_2 = -x_1^2$ and $p_1 = x_2$ from Example 25, where $\psi_2' = (\psi_2 \wedge \varphi)$ implies that it is always non-positive for large enough n. Now we obtain $M_2 = 0$ (the 1-monotonicity threshold of $(16, 0)$ and $(9, 1)$) and $N_2 = 1$, where $C = C_\alpha = \max\{1, N_2, M_2\} = 1$. Moreover, we have $D_\alpha = \max\{1, 0\} = 1$, since

$$\begin{aligned} &1 \text{ is the 1-monotonicity threshold of } (9,0) \text{ and } (1,0), \text{ and}\\ &0 \text{ is the 1-monotonicity threshold of } (1,0) \text{ and } (0,0). \end{aligned}$$

We now get the tighter bound $\mathrm{sth}_{\psi_2}^{\sqcup} = 2 \cdot \sqcup\{p_1\} + \max\{n_0, C_\alpha, D_\alpha\} = 2 \cdot x_2 + 1$ for $t_5 \star t_5$. So t_5's runtime bound is $2 \cdot \mathrm{sth}_{\psi_2}^{\sqcup} + 1 = 4 \cdot x_2 + 3$ when using invariant ψ_2.

Theorem 29 shows how the technique of Lemma 16 and Theorem 27 can be used to compute local runtime bounds for twn-loops whenever such loops occur within an integer program. To this end, one needs the new Theorem 11 where in contrast to [6,18] these local bounds do not have to result from ranking functions.

To turn a self-loop t and $r \in \mathcal{E}_{\{t\}}$ from a larger program $\mathcal{P}$ into a twn-loop (ψ, φ, η), we use t's guard φ and update η. To obtain an update-invariant ψ, our implementation uses the Apron library [23] for computing invariants on a version of the full program where we remove all entry transitions $\mathcal{E}_{\{t\}}$ except r.[5] From the invariants computed for t, we take those that are also update-invariants of t.

Theorem 29 (Local Bounds for Twn-Loops). *Let $\mathcal{P} = (\mathcal{PV}, \mathcal{L}, \ell_0, \mathcal{T})$ be an integer program with $\mathcal{PV}' = \{x_1, \ldots, x_d\} \subseteq \mathcal{PV}$. Let $t = (\ell, \varphi, \eta, \ell) \in \mathcal{T}$ with $\varphi \in \mathcal{F}(\mathcal{PV}')$, $\eta(v) \in \mathbb{Z}[\mathcal{PV}']$ for all $v \in \mathcal{PV}'$, and $\eta(v) = v$ for all $v \in \mathcal{PV} \backslash \mathcal{PV}'$. For any entry transition $r \in \mathcal{E}_{\{t\}}$, let $\psi \in \mathcal{F}(\mathcal{PV}')$ such that $\models \psi \to \eta(\psi)$ and*

[5] Regarding invariants for the full program in the computation of local bounds for t is possible since in contrast to [6,18] our definition of local bounds from Definition 9 is restricted to states that are reachable from an initial configuration (ℓ_0, σ_0).

such that $\sigma(\psi)$ holds whenever there is a $\sigma_0 \in \Sigma$ with $(\ell_0, \sigma_0) \to_{\mathcal{T}}^* \circ \to_r (\ell, \sigma)$. If $L = (\psi, \varphi, \eta)$ is a terminating tnn-loop, then let $\mathcal{RB}_{loc}(\to_r \{t\}) = \mathrm{sth}^{\sqcup}$, where $\mathrm{sth}^{\sqcup}$ is defined as in Theorem 27. If L is a terminating twn-loop but no tnn-loop, let $\mathcal{RB}_{loc}(\to_r \{t\}) = 2 \cdot \mathrm{sth}^{\sqcup} + 1$, where $\mathrm{sth}^{\sqcup}$ is the bound of Theorem 27 computed for $L \star L$. Otherwise, let $\mathcal{RB}_{loc}(\to_r \{t\}) = \omega$. Then $\mathcal{RB}_{loc}$ is a local runtime bound for $\{t\} = \mathcal{T}'_{>} = \mathcal{T}'$ in the program $\mathcal{P}$.

Example 30. In Fig. 1, we consider the self-loop t_5 with $\mathcal{E}_{\{t_5\}} = \{t_4, t_1\}$ and the update-invariants $\psi_1 = \mathtt{true}$ resp. $\psi_2 = (x_3 > 0)$. For t_5's guard φ and update η, both (ψ_i, φ, η) are terminating twn-loops (see Example 14), i.e., (2) is invalid.

By Theorem 29 and Example 28, $\mathcal{RB}_{\mathrm{loc}}$ with $\mathcal{RB}_{\mathrm{loc}}(\to_{t_4} \{t_5\}) = 4 \cdot x_2 + 4 \cdot x_3^3 + 4 \cdot x_3^5 + 3$ and $\mathcal{RB}_{\mathrm{loc}}(\to_{t_1} \{t_5\}) = 4 \cdot x_2 + 3$ is a local runtime bound for $\{t_5\} = \mathcal{T}'_{>} = \mathcal{T}'$ in the program of Fig. 1. As shown in Example 12, Theorem 11 then yields the global runtime bound $\mathcal{RB}_{\mathrm{glo}}(t_5) = 8 \cdot x_4 \cdot x_5 + 13006 \cdot x_4$.

5 Local Runtime Bounds for Twn-Cycles

Section 4 introduced a technique to determine local runtime bounds for twn-self-loops in a program. To increase its applicability, we now extend it to larger cycles. For every entry transition of the cycle, we *chain* the transitions of the cycle, starting with the transition which follows the entry transition. In this way, we obtain loops consisting of a single transition. If the chained loop is a twn-loop, we can apply Theorem 29 to compute a local runtime bound. Any local bound on the chained transition is also a bound on each of the original transitions.[6]

By Theorem 29, we obtain a bound on the number of evaluations of the *complete cycle*. However, we also have to consider a *partial execution* which stops before traversing the full cycle. Therefore, we increase every local runtime bound by 1.

Note that this replacement of a cycle by a self-loop which results from chaining its transitions is only sound for *simple* cycles. A cycle is simple if each iteration through the cycle can only be done in a unique way. So the cycle must not have any subcycles and there also must not be any indeterminisms concerning the next transition to be taken. Formally, $\mathcal{C} = \{t_1, \dots, t_n\} \subset \mathcal{T}$ is a simple cycle if $\mathcal{C}$ does not contain temporary variables and there are pairwise different locations $\ell_1, \dots, \ell_n$ such that $t_i = (\ell_i, \_, \_, \ell_{i+1})$ for $1 \leq i \leq n-1$ and $t_n = (\ell_n, \_, \_, \ell_1)$. This ensures that if there is an evaluation with $\to_{t_i} \circ \to_{\mathcal{C} \setminus \{t_i\}}^* \circ \to_{t_i}$, then the steps with $\to_{\mathcal{C} \setminus \{t_i\}}^*$ have the form $\to_{t_{i+1}} \circ \dots \circ \to_{t_n} \circ \to_{t_1} \circ \dots \circ \to_{t_{i-1}}$.

Algorithm 1 describes how to compute a local runtime bound for a simple cycle $\mathcal{C} = \{t_1, \dots, t_n\}$ as above. In the loop of Line 2, we iterate over all entry transitions r of $\mathcal{C}$. If r reaches the transition t_i, then in Line 3 and 4 we chain $t_i \star \dots \star t_n \star t_1 \star \dots \star t_{i-1}$ which corresponds to one iteration of the cycle starting

[6] This is sufficient for our improved definition of local bounds in Definition 9 where in contrast to [6,18] we do not require a bound on the *sum* but only on *each* transition in the considered set $\mathcal{T}'$. Moreover, here we again benefit from our extension to compute individual local bounds for different entry transitions.

Algorithm 1. Algorithm to Compute Local Runtime Bounds for Cycles

```
   input  : A program $(\mathcal{PV}, \mathcal{L}, \ell_0, \mathcal{T})$ and a simple cycle $\mathcal{C} = \{t_1, \ldots, t_n\} \subset \mathcal{T}$
   output : A local runtime bound $\mathcal{RB}_{\mathrm{loc}}$ for $\mathcal{C} = \mathcal{T}'_{>} = \mathcal{T}'$
1  Initialize $\mathcal{RB}_{\mathrm{loc}}$: $\mathcal{RB}_{\mathrm{loc}}(\rightarrow_r \mathcal{C}) = \omega$ for all $r \in \mathcal{E}_{\mathcal{C}}$.
2  forall $r \in \mathcal{E}_{\mathcal{C}}$ do
3      Let $i \in \{1, \ldots, n\}$ such that $r$'s target location is the start location $\ell_i$ of $t_i$.
4      Let $t = t_i \star \ldots \star t_n \star t_1 \star \ldots \star t_{i-1}$.
5      if there exists a renaming $\pi$ of $\mathcal{PV}$ such that $\pi(t)$ results in a twn-loop then
6          Set $\mathcal{RB}_{\mathrm{loc}}(\rightarrow_r \mathcal{C}) \leftarrow \pi^{-1}(1+$ result of Theorem 29 on $\pi(t)$ and $\pi(r))$.
7  return local runtime bound $\mathcal{RB}_{\mathrm{loc}}$.
```

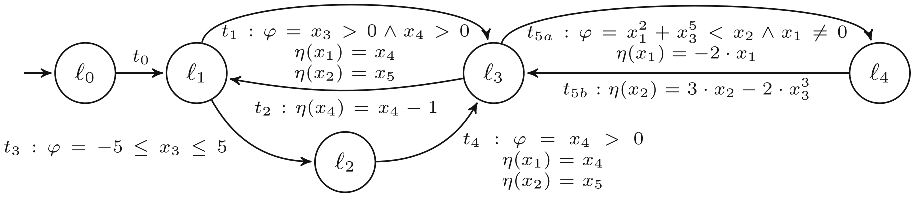

ℓ_0
t_0
ℓ_1
$t_1 : \varphi = x_3 > 0 \wedge x_4 > 0$
$\eta(x_1) = x_4$
$\eta(x_2) = x_5$
$t_2 : \eta(x_4) = x_4 - 1$
$t_3 : \varphi = -5 \leq x_3 \leq 5$
ℓ_2
ℓ_3
$t_4 : \varphi = x_4 > 0$
$\eta(x_1) = x_4$
$\eta(x_2) = x_5$
$t_{5a} : \varphi = x_1^2 + x_3^5 < x_2 \wedge x_1 \neq 0$
$\eta(x_1) = -2 \cdot x_1$
$t_{5b} : \eta(x_2) = 3 \cdot x_2 - 2 \cdot x_3^3$
ℓ_4

Fig. 2. An Integer Program with a Nested Non-Self-Loop

in t_i. If a suitable renaming (and thus also reordering) of the variables turns the chained transition into a twn-loop, then we use Theorem 29 to compute a local runtime bound $\mathcal{RB}_{\mathrm{loc}}(\rightarrow_r \mathcal{C})$ in Lines 5 and 6. If the chained transition does not give rise to a twn-loop, then $\mathcal{RB}_{\mathrm{loc}}(\rightarrow_r \mathcal{C})$ is ω (Line 1). In practice, to use the twn-technique for a transition t in a program, our tool KoAT searches for those simple cycles that contain t and where the chained cycle is a twn-loop. Among those cycles it chooses the one with the smallest runtime bounds for its entry transitions.

Theorem 31 (Correctness of Algorithm 1). *Let $\mathcal{P} = (\mathcal{PV}, \mathcal{L}, \ell_0, \mathcal{T})$ be an integer program and let $\mathcal{C} \subset \mathcal{T}$ be a simple cycle in $\mathcal{P}$. Then the result $\mathcal{RB}_{loc} : \mathcal{E}_{\mathcal{C}} \rightarrow \mathcal{B}$ of Algorithm 1 is a local runtime bound for $\mathcal{C} = \mathcal{T}'_{>} = \mathcal{T}'$.*

Example 32. We apply Algorithm 1 on the cycle $\mathcal{C} = \{t_{5a}, t_{5b}\}$ of the program in Fig. 2. $\mathcal{C}$'s entry transitions t_1 and t_4 both end in ℓ_3. Chaining t_{5a} and t_{5b} yields the transition t_5 of Fig. 1, i.e., $t_5 = t_{5a} \star t_{5b}$. Thus, Algorithm 1 essentially transforms the program of Fig. 2 into Fig. 1. As in Example 28 and 30, we obtain $\mathcal{RB}_{\mathrm{loc}}(\rightarrow_{t_4} \mathcal{C}) = 1 + (2 \cdot \mathrm{sth}^{\sqcup}_{\mathtt{true}} + 1) = 4 \cdot x_2 + 4 \cdot x_3^3 + 4 \cdot x_3^5 + 4$ and $\mathcal{RB}_{\mathrm{loc}}(\rightarrow_{t_1} \mathcal{C}) = 1 + (2 \cdot \mathrm{sth}^{\sqcup}_{x_3>0} + 1) = 4 \cdot x_2 + 4$, resulting in the global runtime bound $\mathcal{RB}_{\mathrm{glo}}(t_{5a}) = \mathcal{RB}_{\mathrm{glo}}(t_{5b}) = 8 \cdot x_4 \cdot x_5 + 13008 \cdot x_4$, which again yields $\mathrm{rc}(\sigma_0) \in \mathcal{O}(n^2)$.

6 Conclusion and Evaluation

We showed that results on subclasses of programs with computable complexity bounds like [19] are not only theoretically interesting, but they have an important practical value. To our knowledge, our paper is the first to integrate such results into an incomplete approach for automated complexity analysis like [6,18]. For this integration, we developed several novel contributions which extend and improve the previous approaches in [6,18,19] substantially:

(a) We extended the concept of local runtime bounds such that they can now depend on entry transitions (Definition 9).
(b) We generalized the computation of global runtime bounds such that one can now lift arbitrary local bounds to global bounds (Theorem 11). In particular, the local bounds might be due to either ranking functions or twn-loops.
(c) We improved the technique for the computation of bounds on twn-loops such that these bounds now take the roles of the different variables into account (Definition 22, Corollary 23, and Theorem 27).
(d) We extended the notion of twn-loops by update-invariants and developed a new over-approximation of their closed forms which takes invariants into account (Definition 13 and 24, Lemma 26, and Theorem 27).
(e) We extended the handling of twn-loops to twn-cycles (Theorem 31).

The need for these improvements is demonstrated by our leading example in Fig. 1 (where the contributions (a)–(d) are needed to infer quadratic runtime complexity) and by the example in Fig. 2 (which illustrates (e)). In this way, the power of automated complexity analysis is increased substantially, because now one can also infer runtime bounds for programs containing non-linear arithmetic.

To demonstrate the power of our approach, we evaluated the integration of our new technique to infer local runtime bounds for twn-cycles in our re-implementation of the tool KoAT (written in OCaml) and compare the results to other state-of-the-art tools. To distinguish our re-implementation of KoAT from the original version of the tool from [6], let KoAT1 refer to the tool from [6] and let KoAT2 refer to our new re-implementation. KoAT2 applies a local control-flow refinement technique [18] (using the tool iRankFinder [8]) and preprocesses the program in the beginning, e.g., by extending the guards of transitions by invariants inferred using the Apron library [23]. For all occurring SMT problems, KoAT2 uses Z3 [28]. We tested the following configurations of KoAT2, which differ in the techniques used for the computation of local runtime bounds:

- KoAT2+RF only uses linear ranking functions to compute local runtime bounds
- KoAT2+MΦRF5 uses multiphase-linear ranking functions of depth ≤ 5
- KoAT2+TWN only uses twn-cycles to compute local runtime bounds (Algorithm 1)
- KoAT2+TWN+RF uses Algorithm 1 for twn-cycles and linear ranking functions
- KoAT2+TWN+MΦRF5 uses Algorithm 1 for twn-cycles and MΦRFs of depth ≤ 5

Existing approaches for automated complexity analysis are already very powerful on programs that only use linear arithmetic in their guards and updates.

	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{>2})$	$\mathcal{O}(EXP)$	$< \infty$	AVG$^+$(s)	AVG(s)
KoAT2 + TWN + MΦRF5	26	231 (5)	73 (5)	13 (4)	1 (1)	344 (15)	8.72	23.93
KoAT2 + TWN + RF	27	227 (5)	73 (5)	13 (4)	1 (1)	341 (15)	8.11	19.77
KoAT2 + MΦRF5	24	226 (1)	68	10	0	328 (1)	8.23	21.63
KoAT2 + RF	25	214 (1)	68	10	1	318 (1)	8.49	16.56
MaxCore	23	216 (2)	66	7	0	312 (2)	2.02	5.31
CoFloCo	22	196 (1)	66	5	0	289 (1)	0.62	2.66
KoAT1	25	169 (1)	74	12	6	286 (1)	1.77	2.77
Loopus	17	170 (1)	49	5 (1)	0	241 (2)	0.42	0.43
KoAT2 + TWN	20 (1)	111 (4)	3 (2)	2 (2)	0	136 (9)	2.54	26.59

Fig. 3. Evaluation on the Collection CINT$^+$

The corresponding benchmarks for *Complexity of Integer Transitions Systems* (CITS) and *Complexity of* C *Integer Programs* (CINT) from the *Termination Problems Data Base* [33] which is used in the annual *Termination and Complexity Competition (TermComp)* [17] contain almost only examples with linear arithmetic. Here, the existing tools already infer finite runtimes for more than 89% of those examples in the collections CITS and CINT where this *might*[7] be possible.

The main benefit of our new integration of the twn-technique is that in this way one can also infer finite runtime bounds for programs that contain non-linear guards or updates. To demonstrate this, we extended both collections CITS and CINT by 20 examples that represent typical such programs, including several benchmarks from the literature [3,14,15,18,20,34], as well as our programs from Fig. 1 and 2. See [27] for a detailed list and description of these examples.

Figure 3 presents our evaluation on the collection CINT$^+$, consisting of the 484 examples from CINT and our 20 additional examples for non-linear arithmetic. We refer to [27] for the (similar) results on the corresponding collection CITS$^+$.

In the C programs of CINT$^+$, all variables are interpreted as integers over $\mathbb{Z}$ (i.e., without overflows). For KoAT2 and KoAT1, we used Clang [7] and llvm2kittel [10] to transform C programs into integer transitions systems as in Definition 2. We compare KoAT2 with KoAT1 [6] and the tools CoFloCo [11,12], MaxCore [2] with CoFloCo in the backend, and Loopus [31]. We do not compare with RaML [21], as it does not support programs whose complexity depends on (possibly negative) integers (see [29]). We also do not compare with PUBS [1], because as stated in [9] by one of its authors, CoFloCo is stronger than PUBS. For the same reason, we only consider MaxCore with the backend CoFloCo instead of PUBS.

All tools were run inside an Ubuntu Docker container on a machine with an AMD Ryzen 7 3700X octa-core CPU and 48 GB of RAM. As in *TermComp*, we applied a timeout of 5 min for every program.

In Fig. 3, the first entry in every cell denotes the number of benchmarks from CINT$^+$ where the respective tool inferred the corresponding bound. The number

[7] The tool LoAT [13,16] proves unbounded runtime for 217 of the 781 examples from CITS and iRankFinder [4,8] proves non-termination for 118 of 484 programs of CINT.

in brackets is the corresponding number of benchmarks when only regarding our 20 new examples for non-linear arithmetic. The runtime bounds computed by the tools are compared asymptotically as functions which depend on the largest initial absolute value n of all program variables. So for instance, there are $26 + 231 = 257$ programs in CINT^+ (and 5 of them come from our new examples) where KoAT2+TWN+MΦRF5 can show that $\mathrm{rc}(\sigma_0) \in \mathcal{O}(n)$ holds for all initial states σ_0 where $|\sigma_0(v)| \leq n$ for all $v \in \mathcal{PV}$. For 26 of these programs, KoAT2+TWN+MΦRF5 can even show that $\mathrm{rc}(\sigma_0) \in \mathcal{O}(1)$, i.e., their runtime complexity is constant. Overall, this configuration succeeds on 344 examples, i.e., "$< \infty$" is the number of examples where a finite bound on the runtime complexity could be computed by the respective tool within the time limit. "AVG$^+$(s)" is the average runtime of the tool on successful runs in seconds, i.e., where the tool inferred a finite time bound before reaching the timeout, whereas "AVG(s)" is the average runtime of the tool on all runs including timeouts.

On the original benchmarks CINT where very few examples contain non-linear arithmetic, integrating TWN into a configuration that already uses multiphase-linear ranking functions does not increase power much: KoAT2+TWN+MΦRF5 succeeds on $344 - 15 = 329$ such programs and KoAT2+MΦRF5 solves $328 - 1 = 327$ examples. On the other hand, if one only has linear ranking functions, then an improvement via our twn-technique has similar effects as an improvement with multiphase-linear ranking functions (here, the success rate of KoAT2+MΦRF5 is similar to KoAT2+TWN+RF which solves $341 - 15 = 326$ such programs).

But the main benefit of our technique is that it also allows to successfully handle examples with non-linear arithmetic. Here, our new technique is significantly more powerful than previous ones. Other tools and configurations without TWN in Fig. 3 solve at most 2 of the 20 new examples. In contrast, KoAT2+TWN+RF and KoAT2+TWN+MΦRF5 both succeed on 15 of them.[8] In particular, our running examples from Fig. 1 and 2 and even isolated twn-loops like t_5 or $t_5 \star t_5$ from Example 14 and 17 can *only* be solved by KoAT2 with our twn-technique.

To summarize, our evaluations show that KoAT2 with the added twn-technique outperforms all other configurations and tools for automated complexity analysis on all considered benchmark sets (i.e., CINT^+, CINT, CITS^+, and CITS) and it is the only tool which is also powerful on examples with non-linear arithmetic.

KoAT's source code, a binary, and a Docker image are available at https://aprove-developers.github.io/KoAT_TWN/. The website also has details on our experiments and *web interfaces* to run KoAT's configurations directly online.

Acknowledgments. We are indebted to M. Hark for many fruitful discussions about complexity, twn-loops, and KoAT. We are grateful to S. Genaim and J. J. Doménech for a suitable version of iRankFinder which we could use for control-flow refinement in KoAT's backend. Moreover, we thank A. Rubio and E. Martín-Martín for a static binary of MaxCore, A. Flores-Montoya and F. Zuleger for help in running CoFloCo and Loopus, F. Frohn for help and advice, and the reviewers for their feedback to improve the paper.

[8] One is the non-terminating leading example of [15], so at most 19 *might* terminate.

References

1. Albert, E., Arenas, P., Genaim, S., Puebla, G.: Automatic inference of upper bounds for recurrence relations in cost analysis. In: Alpuente, M., Vidal, G. (eds.) SAS 2008. LNCS, vol. 5079, pp. 221–237. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69166-2_15
2. Albert, E., Bofill, M., Borralleras, C., Martín-Martín, E., Rubio, A.: Resource analysis driven by (conditional) termination proofs. Theory Pract. Logic Program. **19**, 722–739 (2019). https://doi.org/10.1017/S1471068419000152
3. Ben-Amram, A.M., Genaim, S.: On multiphase-linear ranking functions. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 601–620. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_32
4. Ben-Amram, A.M., Doménech, J.J., Genaim, S.: Multiphase-linear ranking functions and their relation to recurrent sets. In: Chang, B.-Y.E. (ed.) SAS 2019. LNCS, vol. 11822, pp. 459–480. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32304-2_22
5. Braverman, M.: Termination of integer linear programs. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 372–385. Springer, Heidelberg (2006). https://doi.org/10.1007/11817963_34
6. Brockschmidt, M., Emmes, F., Falke, S., Fuhs, C., Giesl, J.: Analyzing runtime and size complexity of integer programs. ACM Trans. Program. Lang. Syst. **38**, 1–50 (2016). https://doi.org/10.1145/2866575
7. Clang Compiler. https://clang.llvm.org/
8. Doménech, J.J., Genaim, S.: iRankFinder. In: Lucas, S. (ed.) WST 2018, p. 83 (2018). http://wst2018.webs.upv.es/wst2018proceedings.pdf
9. Doménech, J.J., Gallagher, J.P., Genaim, S.: Control-flow refinement by partial evaluation, and its application to termination and cost analysis. Theory Pract. Logic Program. **19**, 990–1005 (2019). https://doi.org/10.1017/S1471068419000310
10. Falke, S., Kapur, D., Sinz, C.: Termination analysis of C programs using compiler intermediate languages. In: Schmidt-Schauß, M. (ed.) RTA 2011. LIPIcs, vol. 10, pp. 41–50 (2011). https://doi.org/10.4230/LIPIcs.RTA.2011.41
11. Flores-Montoya, A., Hähnle, R.: Resource analysis of complex programs with cost equations. In: Garrigue, J. (ed.) APLAS 2014. LNCS, vol. 8858, pp. 275–295. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12736-1_15
12. Flores-Montoya, A.: Upper and lower amortized cost bounds of programs expressed as cost relations. In: Fitzgerald, J., Heitmeyer, C., Gnesi, S., Philippou, A. (eds.) FM 2016. LNCS, vol. 9995, pp. 254–273. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48989-6_16
13. Frohn, F., Giesl, J.: Proving non-termination via loop acceleration. In: Barrett, C.W., Yang, J. (eds.) FMCAD 2019, pp. 221–230 (2019). https://doi.org/10.23919/FMCAD.2019.8894271
14. Frohn, F., Giesl, J.: Termination of triangular integer loops is decidable. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11562, pp. 426–444. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25543-5_24
15. Frohn, F., Hark, M., Giesl, J.: Termination of polynomial loops. In: Pichardie, D., Sighireanu, M. (eds.) SAS 2020. LNCS, vol. 12389, pp. 89–112. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65474-0_5, https://arxiv.org/abs/1910.11588
16. Frohn, F., Naaf, M., Brockschmidt, M., Giesl, J.: Inferring lower runtime bounds for integer programs. ACM Trans. Program. Lang. Syst. **42**, 1–50 (2020). https://doi.org/10.1145/3410331

17. Giesl, J., Rubio, A., Sternagel, C., Waldmann, J., Yamada, A.: The termination and complexity competition. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) TACAS 2019. LNCS, vol. 11429, pp. 156–166. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17502-3_10
18. Giesl, J., Lommen, N., Hark, M., Meyer, F.: Improving automatic complexity analysis of integer programs. In: The Logic of Software: A Tasting Menu of Formal Methods. LNCS, vol. 13360 (to appear). Also appeared in CoRR, abs/2202.01769. https://arxiv.org/abs/2202.01769
19. Hark, M., Frohn, F., Giesl, J.: Polynomial loops: beyond termination. In: Albert, E., Kovács, L. (eds.) LPAR 2020, EPiC, vol. 73, pp. 279–297 (2020). https://doi.org/10.29007/nxv1
20. Heizmann, M., Leike, J.: Ranking templates for linear loops. Log. Methods Comput. Sci. **11**(1), 16 (2015). https://doi.org/10.2168/LMCS-11(1:16)2015
21. Hoffmann, J., Das, A., Weng, S.-C.: Towards automatic resource bound analysis for OCaml. In: Castagna, G., Gordon, A.D. (eds.) POPL 2017, pp. 359–373 (2017). https://doi.org/10.1145/3009837.3009842
22. Hosseini, M., Ouaknine, J., Worrell, J.: Termination of linear loops over the integers. In: Baier, C., Chatzigiannakis, I., Flocchini, P., Leonardi, S. (eds.) ICALP 2019, LIPIcs, vol. 132 (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.118
23. Jeannet, B., Miné, A.: Apron: a library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 661–667. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_52
24. Kincaid, Z., Breck, J., Cyphert, J., Reps, T.W.: Closed forms for numerical loops. Proc. ACM Program. Lang. **3**(POPL), 1–29 (2019). https://doi.org/10.1145/3290368
25. Kovács, L.: Reasoning algebraically about p-solvable loops. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 249–264. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_18
26. Lommen, N., Meyer, F., Giesl, J.: Automatic complexity analysis of integer programs via triangular weakly non-linear loops. CoRR abs/2205.08869 (2022). https://arxiv.org/abs/2205.08869
27. Lommen, N., Meyer, F., Giesl, J.: Empirical evaluation of: "Automatic complexity analysis of integer programs via triangular weakly non-linear loops". https://aprove-developers.github.io/KoAT_TWN/
28. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
29. RaML (Resource Aware ML). https://www.raml.co/interface/
30. Rodríguez-Carbonell, E., Kapur, D.: Automatic generation of polynomial loop invariants: algebraic foundation. In: Gutierrez, J. (ed.) ISSAC 2004, pp. 266–273 (2004). https://doi.org/10.1145/1005285.1005324
31. Sinn, M., Zuleger, F., Veith, H.: Complexity and resource bound analysis of imperative programs using difference constraints. J. Autom. Reason. **59**, 3–45 (2017). https://doi.org/10.1007/s10817-016-9402-4
32. Tiwari, A.: Termination of linear programs. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 70–82. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_6
33. TPDB (Termination Problems Data Base). https://github.com/TermCOMP/TPDB
34. Xu, M., Li, Z.-B.: Symbolic termination analysis of solvable loops. J. Symb. Comput. **50**, 28–49 (2013). https://doi.org/10.1016/j.jsc.2012.05.005

Author Index