and Viktor Kuncak, editors, *Proceedings of the 29th International Conference on Computer Aided Verification*, volume 10427 of *Lecture Notes in Computer Science*, pages 592–600, Heidelberg, Germany, July 2017. Springer-Verlag.

15. Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for action-labelled quantitative transition systems. In Antonio Cerone and Herbert Wiklicky, editors, *Proceedings of 3rd Workshop on Quantitative Aspects of Programming Languages*, volume 153(2) of *Electronic Notes in Theoretical Computer Science*, pages 79–96, Edinburgh, UK, April 2005. Elsevier.
16. Yuxin Deng and Rob van Glabbeek. Characterising probabilistic processes logically. In Christian G. Fermüller and Andrei Voronkov, editors, *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6397 of *Lecture Notes in Computer Science*, pages 278–293, Yogyakarta, Indonesia, October 2010. Springer-Verlag.
17. Salem Derisavi, Holger Hermanns, and William Sanders. Optimal state-space lumping in Markov chains. *Information Processing Letters*, 87(6):309–315, September 2003.
18. Josée Desharnais. *Labelled Markov Processes*. PhD thesis, McGill University, Montreal, November 1999.
19. Josée Desharnais, Abbas Edalat, and Prakash Panangaden. A logical characterization of bisimulation for labeled Markov processes. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 478–487, Indianapolis, IN, USA, June 1998. IEEE.
20. Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labeled Markov systems. In Jos Baeten and Sjouke Mauw, editors, *Proceedings of the 10th International Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 258–273, Eindhoven, The Netherlands, August 1999. Springer-Verlag.
21. Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, June 2004.
22. Josée Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 413–422, Copenhagen, Denmark, July 2002. IEEE.
23. Josée Desharnais, François Laviolette, and Mathieu Tracol. Approximate analysis of probabilistic processes: logic, simulation and games. In *Proceedings of the 5th International Conference on the Quantitative Evaluation of Systems*, pages 264–273, Saint-Malo, France, September 2008. IEEE.
24. Wenjie Du, Yuxin Deng, and Daniel Gebler. Behavioural pseudometrics for nondeterministic probabilistic systems. In Martin Fränzle, Deepak Kapur, and Naijun Zhan, editors, *Proceedings of the 2nd International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, volume 9984 of *Lecture Notes in Computer Science*, pages 67–84, Beijing, China, November 2016. Springer-Verlag.
25. Yuan Feng and Lijun Zhang. When equivalence and bisimulation join forces in probabilistic automata. In Cliff Jones, Pekka Pihlajasaari, and Jun Sun, editors, *Proceedings of the 19th International Symposium on Formal Methods*, volume 8442 of *Lecture Notes in Computer Science*, pages 247–262, Singapore, May 2014. Springer-Verlag.

26. Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov decision processes. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence*, pages 162–169, Banff, Canada, July 2004. AUAI Press.
27. Alessandro Giacalone, Chi-Chang Jou, and Scott Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the IFIP WG 2.2/2.3 Working Conference on Programming Concepts and Methods*, pages 443–458, Sea of Gallilee, Israel, April 1990. North-Holland.
28. Susanne Graf and Joseph Sifakis. A modal characterization of observational congruence on finite terms of CCS. In Jan Paredaens, editor, *Proceedings of the 11th Colloquium on Automata, Languages and Programming*, volume 172 of *Lecture Notes in Computer Science*, pages 222–234, Antwerp, Belgium, July 1984. Springer-Verlag.
29. Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In Jaco de Bakker and Jan van Leeuwen, editors, *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309, Noordwijkerhout, The Netherlands, July 1980. Springer-Verlag.
30. Michael Hillerström. Verification of CSS-processes. Master's thesis, Aalborg University, Aalborg, Denmark, January 1987.
31. Leonid Kantorovich and Gennadi Rubinstein. On the space of completely additive functions (in Russian). *Vestnik Leningradskogo Universiteta*, 3(2):52–59, 1958.
32. Leonid Khachiyan. A polynomial algorithm in linear programming (in Russian). *Doklady Akademii Nauk SSSR*, 244(5):1093–1096, 1979. English translation in *Soviet Mathematics Doklady*, 20:191–194, 1979.
33. Viktor Klee and Christoph Witzgall. Facets and vertices of transportation polytopes. In George Dantzig and Arthur Veinott, editors, *Proceedings of 5th Summer Seminar on the Mathematics of the Decision Sciences*, volume 11 of *Lectures in Applied Mathematics*, pages 257–282, Stanford, CA, USA, June/July 1967. AMS.
34. Stephen Kleene. *Introduction to Metamathematics*. Van Nostrand, New York, NY, USA, 1952.
35. Yuichi Komorida, Shin-ya Katsumata, Clemens Kupke, Jurriaan Rot, and Ichiro Hasuo. Expressivity of quantitative modal logics : Categorical foundations via codensity and approximation. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–14, Rome, Italy, June/July 2021.
36. Barbara König and Christina Mika-Michalski. (Metric) bisimulation games and real-valued modal logics for coalgebras. In Sven Schewe and Lijun Zhang, editors, *Proceedings of the 29th International Conference on Concurrency Theory*, volume 118 of *Leibniz International Proceedings in Informatics*, pages 37:1–37:17, Beijing, China, September 2018. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.
37. Barbara König, Christina Mika-Michalski, and Lutz Schröder. Explaining nonbisimilarity in a coalgebraic approach: Games and distinguishing formulas. In Daniela Petrisan and Jurriaan Rot, editors, *Proceedings of 15th IFIP WG 1.3 International Workshop on Coalgebraic Methods in Computer Science*, volume 12094 of *Lecture Notes in Computer Science*, pages 133–154, Dublin, Ireland, April 2020. Springer-Verlag.
38. Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591, Snowbird, UT, USA, July 2011. Springer-Verlag.

39. Kim Larsen and Arne Skou. Bisimulation through probabilistic testing. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages*, pages 344–352, Austin, TX, USA, January 1989. ACM.
40. David Luenberger and Yinyu Ye. *Linear and nonlinear programming*. Springer-Verlag, New York, NY, USA, 2008.
41. Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1980.
42. James Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78(2):109–129, August 1997.
43. Vera Pantelic and Mark Lawford. A pseudometric in supervisory control of probabilistic discrete event systems. *Discrete Event Dynamic Systems*, 22(4):479–510, December 2012.
44. David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Proceedings of 5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, Karlsruhe, Germany, March 1981. Springer-Verlag.
45. Amgad Rady and Franck van Breugel. Java code to explain probabilistic bisimilarity distances for labelled Markov chains, February 2023. https://doi.org/10.5281/zenodo.7626542.
46. Michel Reniers, Rob Schoren, and Tim Willemse. Results on embeddings between state-based and event-based systems. *The Computer Journal*, 57(1):73–92, 2014.
47. Joshua Sack and Lijun Zhang. A general framework for probabilistic characterizing formulae. In Viktor Kuncak and Andrey Rybalchenko, editors, *Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 7148 of *Lecture Notes in Computeer Science*, pages 396–411, Philadelphia, PA, USA, January 2012. Springer-Verlag.
48. Qiyi Tang. *Computing probabilistic bisimilarity distances*. PhD thesis, York University, Toronto, Canada, August 2018.
49. Kathleen Trustrum. *Linear programming*. Routledge & Kegan Paul, London, UK, 1971.
50. Antti Valmari and Giuliana Franceschinis. Simple $O(m \log n)$ time Markov chain lumping. In Javier Esparza and Rupak Majumdar, editors, *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 38–52, Paphos, Cyprus, March 2010. Springer-Verlag.
51. Paul Wild and Lutz Schröder. Characteristic logics for behavioural metrics via fuzzy lax extensions. In Igor Konnov and Laura Kovács, editors, *Proceedings of the 31st International Conference on Concurrency Theory*, volume 171 of *Leibniz International Proceedings in Informatics*, pages 27:1–27:23, Vienna, Austria, September 2020. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.
52. Thorsten Wißmann, Stefan Milius, and Lutz Schröder. Explaining behavioural inequivalence generically in quasilinear time. In Serge Haddad and Daniele Varacca, editors, *Proceedings of the 32nd International Conference on Concurrency Theory*, volume 203 of *Leibniz International Proceedings in Informatics*, pages 32:1–32:18, Paris, France, April 2021. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Weighted and Branching Bisimilarities from Generalized Open Maps

Jérémy Dubut[1](✉) and Thorsten Wißmann[2,3]*

[1] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
jeremy.dubut@aist.go.jp
[2] Radboud University, Nijmegen, the Netherlands
t.wissmann@cs.ru.nl
[3] Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany

Abstract. In the open map approach to bisimilarity, the paths and their runs in a given state-based system are the first-class citizens, and bisimilarity becomes a derived notion. While open maps were successfully used to model bisimilarity in non-deterministic systems, the approach fails to describe quantitative system equivalences such as probabilistic bisimilarity. In the present work, we see that this is indeed impossible and we thus generalize the notion of open maps to also accommodate weighted and probabilistic bisimilarity. Also, extending the notions of strong path and path bisimulations into this new framework, we show that branching bisimilarity can be captured by this extended theory and that it can be viewed as the history preserving restriction of weak bisimilarity.

Keywords: Open maps · Weighted Bisimilarity · Probabilistic Bisimilarity · Branching Bisimilarity · Weak Bisimilarity

1 Introduction

The theory of open maps is a categorical framework to reason about systems and their bisimilarities [16]. Given a category of systems and a description of the shape of the executions and how to extend them, open maps are morphisms with lifting properties with respect to those extensions. Intuitively, open maps are morphisms which preserve and reflect transitions of systems, that is, they are morphisms whose graphs are bisimulations. The theory covers various classical notions of bisimilarity. For example, two LTSs are strongly bisimilar if and only if there is a span of open maps between them. Varying the category of models and the execution shapes allows describing weak bisimilarity, timed bisimilarity, probabilistic Larsen and Skou bisimilarity, and history-preserving bisimilarity of event structures (see [16,3,12] for examples).

Another categorical framework for bisimilarity is coalgebra [22]. This time, given a category and an endofunctor describing respectively the type of state spaces and the type of transitions, a 'system' is understood as a coalgebra for this

* Supported by the NWO TOP project 612.001.852.

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 308–327, 2023.
https://doi.org/10.1007/978-3-031-30829-1_15

functor. Coalgebra homomorphisms are then very similar to open maps in spirit: they also are morphisms that preserve and reflect transitions. This intuition has been made formal by transformations between the categorical frameworks in both ways; from open maps to coalgebra [19], and conversely [25]. However, the latter suggests that open maps are only adapted to modeling non-deterministic systems and would struggle with other types of branchings, such as probabilistic.

In coalgebra, there are no particular difficulties in modeling weighted systems, and by extension, discrete probabilistic systems [17]. There is also some work for continuous probabilities, although the theory is much more complicated [5,4]. As we will explain more precisely later, there have been some attempts to do so with open maps in [3,5], but the result is somewhat disappointing.

Conversely, coalgebra is not adapted to bisimilarities for systems where transitions are not history-preserving, that is, for which the behavioral equivalence does not just depend on the transitions at a given state, but on the whole history of the execution that led to this state. That is the case for example for branching bisimilarity [23]. Branching bisimilarity arose precisely to make weak bisimilarity history-preserving. In [3], weak bisimilarity has been described using open maps by carefully choosing the underlying category, with a general theory developed in [9] using presheaf models. Branching bisimilarity has also been studied using open maps in [1,2], but indirectly, through a translation into presheaves.

To resume, the goal of this paper is to capture weighted and branching bisimilarities using a generalization of open maps. Concretely, the contributions are:

1. a proof that it is impossible to appropriately model probabilistic system using standard open maps (Section 3.2),
2. a faithful extension of the theory of open maps and (strong) path bisimulations (Section 4),
3. a generalized open map situation capturing weighted and probabilistic bisimilarities (Section 5),
4. a generalized open map situation where strong path bisimulations correspond to stuttering branching bisimulations, open map bisimilarity to branching bisimilarity, and path bisimulations to weak bisimulations (Section 6).

Full proofs can be found in the appendix: http://arxiv.org/abs/2301.07004

2 From Path Categories to Bisimilarity

Before discussing weighted bisimilarity, let us first recall the main ideas of modeling bisimilarity via open maps, as introduced by Joyal et al. [16]. The definition is parametric in a functor $J\colon \mathbb{P} \to \mathbb{M}$, from a category $\mathbb{P}$ of paths to a category $\mathbb{M}$ of models or systems of interest. In the prime example, $\mathbb{M}$ is the category of labelled transition systems LTS as defined next:

Definition 2.1. *For a fixed set A of labels, the category LTS contains:*

1. *Objects: a labelled transition system $(X, \rightarrow, x_0)$ is a set X of states, a transition relation $\rightarrow \subseteq X \times A \times X$ and a distinguished initial state $x_0 \in X$. We*

write $x \xrightarrow{a} x'$ *to denote that* $(x, a, x') \in \to$ *and simply refer to the LTS as* X *if* $\to$ *and* x_0 *are clear from the context. For disambiguation, we use* $\to$ *for morphisms and* $\to$ *for transitions.*

2. *Morphisms: a* functional simulation $f\colon (X, \to, x_0) \to (Y, \to, y_0)$ *is a function* $f\colon X \to Y$ *with* $f(x_0) = y_0$ *and for all* $x \xrightarrow{a} x'$ *in* X*, we have* $f(x) \xrightarrow{a} f(x')$*.*

A functional simulation $f\colon X \to Y$ intuitively means that the system Y has at least the transitions of X, but possibly more. A special case of a functional simulation is the *run* of a word in a system:

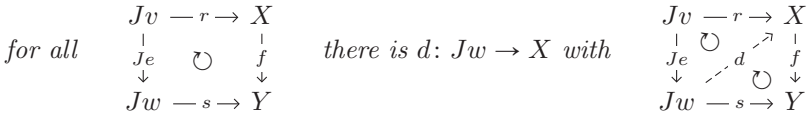
Definition 2.2. *For the label set* A*, let* $(A^*, \leqslant)$ *be the partially ordered set of words, ordered by the prefix ordering. The functor* $J\colon (A^*, \leqslant) \to \mathsf{LTS}$ *sends a word* $w \in A^*$ *to the LTS* $Jw = (\{v \mid v \leqslant w\}, \to, \varepsilon)$ *of all prefixes of* w *with* $v \xrightarrow{a} va$ *for all* $a \in A$*,* $va \leqslant w$*.*

This functor J (or more precisely, its image) is often called *path category* of LTS: the possible runs of a word $w \in A^*$ in $(X, \to, x_0)$ correspond precisely to the functional simulations $Jw \to (X, \to, x_0)$ in LTS.

On the abstract level, for a general functor $J\colon \mathbb{P} \to \mathbb{M}$, we understand the set of morphisms $r\colon Jw \to X$ for $w \in \mathbb{P}$ and $X \in \mathbb{M}$ as the runs of the path w in the model X. We can already make the trivial observation that all morphisms $f\colon X \to Y$ in $\mathbb{M}$ preserve runs: given a run $r\colon Jw \to X$ of some path $w \in \mathbb{P}$ in X, there is a run $f \cdot r\colon Jw \to Y$ of w in Y.

The converse does not hold for a general $f\colon X \to Y$ in $\mathbb{M}$: given a run of w in Y, there is not necessarily a run of w in X. If f reflects runs, it is called *open*:

Definition 2.3. *For a functor* $J\colon \mathbb{P} \to \mathbb{M}$*, a morphism* $f\colon X \to Y$ *in* $\mathbb{M}$ *is called* open *if* f *satisfies the following* lifting property *for all* $e\colon v \to w$ *in* $\mathbb{P}$*:*



That is, for all commutative squares ($s \cdot Je = f \cdot r$), there is $d\colon Jw \to X$ in $\mathbb{M}$ that makes both triangles on the right commute ($f \cdot d = s$ and $d \cdot Je = r$).

By construction, we can only make statements about states that are reachable via some run. Thus, one often restricts $\mathbb{M}$ beforehand to contain only models in which all states are reachable from the initial state.

For LTSs in which all states are reachable from the initial state, open maps are related to strong bisimulations [20]: open maps are precisely functions whose graph relation $\{(x, fx) \mid x \in X\}$ is a strong bisimulation. Reformulated in the context of allegories [10], open maps are precisely the maps in the allegory of relations that are strong bisimulations. It is then natural to recover bisimulations as tabulations of open maps, that is:

Definition 2.4. *For a functor* $J\colon \mathbb{P} \to \mathbb{M}$*, we say that two models* X *and* Y *are* J-bisimilar*, if there exist another model* Z *and two* J*-open maps* $f\colon Z \to X$ *and* $g\colon Z \to Y$*, that is, if there is a span of* J*-open maps between them.*

Of course, J-bisimilarity is a reflexive (identities are open maps) and symmetric (by permuting f and g in the definition) relation on models, but it is not transitive in general. It is when the category $\mathbb{M}$ has pullbacks [16].

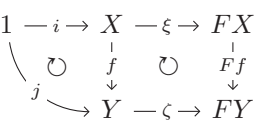
Given a functor $J\colon \mathbb{P} \to \mathbb{M}$, there are more classical ways of defining bisimilarities given in [16]. The first one is *(strong) path bisimulations*, which are relations on runs (similar to history-preserving bisimulations) satisfying the usual bisimilarity conditions. The second one is by using a modal logic similar to the Hennessy-Milner theorem. In the case of LTSs with strong bisimilarity, all those notions describe the same notion of bisimilarity, but that is not true for general $J\colon \mathbb{P} \to \mathbb{M}$: it can only be proved that J-bisimilarity implies the existence of a (strong) path bisimulation, which itself implies that the two models satisfy the same formulas of the modal logic. In [6], some mild sufficient conditions in terms of trees (i.e., colimits of paths in $\mathbb{M}$) are given for those three notions to coincide. In particular, all the examples of bisimilarities covered by open maps cited earlier satisfy these conditions.

We use coalgebra for uniform statements about state-based systems of different branching type (including non-deterministic and probabilistic branching):

Definition 2.5. *For an object* 1 *of a category* $\mathcal{C}$ *and an endofunctor* $F\colon \mathcal{C} \to \mathcal{C}$, *a* pointed coalgebra *is a pair of morphisms of* $\mathcal{C}$ *of the form* $1 \xrightarrow{i} X \xrightarrow{\xi} FX$.

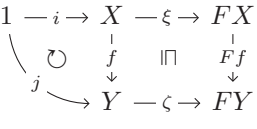
For example, LTSs can be modeled as pointed coalgebras with $\mathcal{C} = \mathsf{Set}$, 1 any singleton, and $F = \mathcal{P}(A \times \_\_)$, where $\mathcal{P}$ is the power set functor. The usual notion of morphisms of coalgebras can be spelt out as follows:

Definition 2.6. *A* (proper) homomorphism *of pointed coalgebras from* (X, ξ, i) *to* (Y, ζ, j) *is a morphism* $f\colon X \to Y$ *of* $\mathcal{C}$ *such that the diagram on the right commutes.*



Pointed coalgebras and proper homomorphisms always form a category, but in the case of LTSs as described above, this category is not equivalent to the category LTS. Indeed, proper homomorphisms are not just morphisms that preserve transitions, but similarly to open maps, they also reflect them. In [25], the authors proved that for a large class of endofunctors, whose coalgebras basically are non-deterministic, proper homomorphisms precisely correspond to J-open maps for a certain functor J. To model morphisms that are only required to preserve transitions, homomorphisms have to be made lax as follows (see [25]):

Definition 2.7. *Assume a relation* $\sqsubseteq$ *on every Hom-set* $\mathcal{C}(X, FY)$. *A* lax homomorphism *of pointed coalgebras from* (X, ξ, i) *to* (Y, ζ, j) *is a morphism* $f\colon X \to Y$ *of* $\mathcal{C}$ *such that the diagram on the right laxly commutes, that is,* $f \cdot i = j$ *and* $Ff \cdot \xi \sqsubseteq \zeta \cdot f$ *in* $\mathcal{C}(X, FY)$.



In the case of the functor $\mathcal{P}(A \times \_\_)$, we can consider the pointwise inclusion on every Hom-set $\mathsf{Set}(X, \mathcal{P}(A \times Y))$. With this, pointed coalgebras and lax

homomorphisms form a category which is isomorphic to the category LTS. However, it is not true in general that they form a category, as a compatibility of $\sqsubseteq$ with the composition is needed as follows:

Definition 2.8. *A partial order on F is a collection of partial orders $\sqsubseteq$, one for each Hom-set of the form $\mathcal{C}(X, FY)$ such that*

$$\forall X \xrightarrow{f_1,f_2} FY,\ X' \xrightarrow{g} X,\ Y \xrightarrow{h} Y' \colon \quad f_1 \sqsubseteq f_2 \quad \Rightarrow \quad Fh \cdot f_1 \cdot g \sqsubseteq Fh \cdot f_2 \cdot g.$$

This is equivalent to the requirement that the Hom-functor $\mathcal{C}(\_, F\_)$ factors through partially ordered sets: $\mathcal{C}(\_, F\_) \colon \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathsf{Pos}$.

Remark 2.9. The present definition subsumes the definition of order on a Set-functor established by Hughes and Jacobs [11, Def 2.1] (details in the appendix).

Lemma 2.10 [25]. *When $\sqsubseteq$ is a partial order on F, pointed coalgebras and lax homomorphisms form a category, which we denote by $\mathsf{LCoalg}(1, F)$.*

Much as with open maps, many flavors of bisimilarity can be recovered using spans of proper homomorphisms:

Definition 2.11. *We say that two pointed coalgebras are coalgebraically bisimilar if there is a span of proper homomorphisms between them.*

There are many ways of defining bisimilarities in coalgebra (see [13] for an overview), but they coincide for the purpose of the present paper.

3 Weighted Bisimilarity and Open Maps

In this section, we describe known attempts to model weighted systems, and particularly probabilistic ones, using open maps. They all work with some variations of the (discrete) distribution functor on Set. We will denote this functor, which maps a set X to the set

$$\mathcal{D}X = \left\{ f \colon X \to [0,1] \mid f^{-1}\big((0,1]\big) \text{ is finite and } \sum_{x \in X} f(x) = 1 \right\},$$

by $\mathcal{D}$ and the variation where the condition $= 1$ is replaced by $\leqslant 1$ by $\mathcal{D}_{\leqslant 1}$ (i.e. $\mathcal{D}_{\leqslant 1}X := \mathcal{D}(X+1)$). We will prove that, even though Larsen-Skou bisimulations for reactive systems can be modeled with open maps, that is impossible for bisimulations for generative systems.

3.1 Larsen-Skou Bisimilarity Using Open Maps

In [3], Cheng et al. describe an open map situation for Probabilistic Transition Systems (PTSs), which corresponds to coalgebras for the functor $(\mathcal{D}(\_) + 1)^A$. In this setting, they consider Partial PTSs (PPTS) which are coalgebras for $(\mathcal{D}^{\varepsilon}_{\leqslant 1}(\_) + 1)^A$ where the sub-probability distributions can have values in hyperreals, allowing infinitesimals ε. The category of PTSs embeds in that of PPTSs,

and the path category is the full subcategory of PPTSs consisting of finite linear systems whose probabilities of transitions are infinitesimals. It is then proved that J-bisimilarity, restricted to PTSs, for this path category corresponds to Larsen-Skou's probabilistic bisimilarity [18].

This open map situation has been reformulated in [7] in terms of coreflections: the obvious functor from PPTSs to TSs is a coreflection whose left-adjoint maps a LTS T to the PPTS whose underlying LTS is T and where all transitions have infinitesimal probabilities. In general, given a coreflection $F : \mathcal{C} \to \mathcal{D}$ with left-adjoint G and a path category J on $\mathcal{D}$, one automatically has the path category $G \circ J$ on $\mathcal{C}$, and this construction preserves good properties of J. In particular, one has that two systems A and B are $(G \circ J)$-bisimilar if and only if FA and FB are J-bisimilar. Cheng et al.'s path category is obtained in this manner with the coreflection above and the standard path category on LTSs. In particular, it means that two PPTSs are bisimilar if and only if their underlying TSs are strongly bisimilar.

3.2 Impossibility Result for Generative Systems

In [5], Desharnais et al. describe several bisimilarities for generative probabilistic systems, that is, coalgebras for the functor $\mathcal{D}_{\leqslant 1}(A \times \_)$, in a coalgebraic way. They pointed out that their efforts to model those bisimilarities using open maps failed [5, p. 188]. In the following, we see that it is in fact not possible. We will show that for generative probabilistic systems modeled by the category $M := \mathsf{LCoalg}(1, \mathcal{D}_{\leqslant 1}(A \times \_))$, there is no open map characterization of the coalgebraic bisimilarity. Actually, the argument here is valid for many other types of weights and is not limited to reals.

Here, for two functions $f, g \colon X \to \mathcal{D}_{\leqslant 1}(Y)$, $f \sqsubseteq g$ means that for all $x \in X$, for all $y \in Y$, $f(x)(y) \leqslant g(x)(y)$, where $\leqslant$ is the usual ordering on $[0, 1]$.

In this situation:

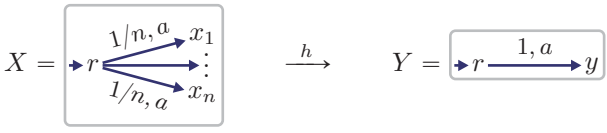
Theorem 3.1. *For $\mathbb{M} := \mathsf{LCoalg}(1, \mathcal{D}_{\leqslant 1}(A \times \_))$ there is no category $\mathbb{P}$ and no functor $J \colon \mathbb{P} \to \mathbb{M}$ such that for every $h \colon X \to Y$ with reachable X the following equivalence holds:*

$$h \text{ is } J\text{-open} \iff h \text{ is a proper homomorphism}$$

and there is no $\mathbb{P}$ and no functor J such that for every X and Y:

$$X \text{ and } Y \text{ are } J\text{-bisimilar} \iff X \text{ and } Y \text{ are coalgebraically bisimilar.}$$

Proof (Sketch). By contradiction, assume that there is such a J. We prove that there is a proper homomorphism of the form:



which cannot be J-open. Consider first the unique lax homomorphism $0_{\mathbb{M}} \to Y$ where $0_{\mathbb{M}}$ consists in one state and no transition. This is not a proper homomorphism, so it is not open by assumption. That is there is a square:

$$\begin{array}{ccc} JP & \xrightarrow{p} & 0_{\mathbb{M}} \\ {\scriptstyle J\phi}\downarrow & & \downarrow{\scriptstyle !_Y} \\ JQ & \xrightarrow{q} & Y \end{array}$$

with no lifting. It is mechanical to check that $JP \simeq 0_{\mathbb{M}}$ and JQ has at least one transition from its initial state to another state $r \xrightarrow{w,a} z$ with $w \neq 0$. With $n = 2 \cdot \lceil \frac{1}{w} \rceil$, the proper homomorphism h above is not open: there cannot be a morphism from JQ to X because $w > \frac{1}{n}$. □

4 Generalized Open Maps

The main argument of the proof of impossibility is the fact that sometimes, a transition with some probability w in the codomain comes from probabilities $w_1, \dots, w_n$ with $\sum_i w_i = w$ in the domain, which makes a lifting morphism impossible with the current framework of open maps.

In this section, we will extend the open map framework with the main intuition that the lifting morphism *splits* the probability w into smaller parts $w_1, \dots, w_n$. After defining these generalized open maps, we show some basic properties of the bisimilarity generated by them.

4.1 Generalized Open Maps Situation

Here, we describe our extension of the open maps framework. The data is similar: we start with a category of models $\mathbb{M}$, but we need more than just a functor $J \colon \mathbb{P} \to \mathbb{M}$. Assume:

- a set V together with a function $J \colon V \to \mathrm{ob}(\mathbb{M})$,
- two small categories $\mathbb{E}$ and $\mathbb{S}$ whose sets of objects are V,
- two functors $J_{\mathbb{E}} \colon \mathbb{E} \to \mathbb{M}$ and $J_{\mathbb{S}} \colon \mathbb{S} \to \mathbb{M}$ coinciding with J on objects.

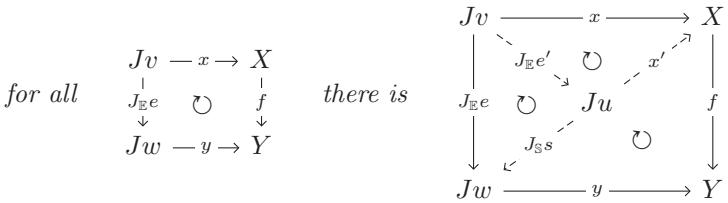
The classical open maps situation $J \colon \mathbb{P} \to \mathbb{M}$ fits in this extension as follows. The category $\mathbb{E}$ is given by $\mathbb{P}$ with the intention that they model path shapes and their *extensions*. The functor $J_{\mathbb{E}}$ is given by J. The category $\mathbb{S}$ is given by the discrete category $|\mathbb{P}|$, that is, the category whose objects are those of $\mathbb{P}$ and whose morphisms are only identities. The functor $J_{\mathbb{S}}$ is the only possible one respecting the conditions of the definition above.

In the general context of this extension, the interpretation is a bit different. Now V is meant to be a set of trees labelled by alphabets and weights. $\mathbb{E}$ still consists in extensions, extending trees into trees with longer branches. $\mathbb{S}$ then consists in *merging morphisms*, similar to the description above: for the example of weighted systems, those morphisms are allowed to merge states into one,

as long as they sum up the weights of the in-going branches. Generally, those morphisms are allowed to perform some merges that are harmless for bisimilarity.

With this data, we can define generalized open maps:

Definition 4.1. *A morphism $f\colon X \to Y$ in $\mathbb{M}$ is called* ($\mathbb{E}$,$\mathbb{S}$)-open *if it satisfies the following* lifting property *for all $e\colon v \to w$ in $\mathbb{E}$:*



The interpretation starts the same as in usual open maps. Assume that we have a tree y in Y extending the image by f of the tree x in X. If f is open, there should be a tree x' extending x and whose image by f is y. However, x' may have a different shape than y, since it might be necessary to split transitions. That is what u and s are modeling: w is obtained from u by merging some states.

The connection with the classical open maps can be formulated as follows

Proposition 4.2. *Given a functor $J\colon \mathbb{P} \to \mathbb{M}$ and a morphism $f\colon X \to Y$,*

$$f \text{ is } J\text{-open if and only if } f \text{ is } (\mathbb{P}, |\mathbb{P}|)\text{-open.}$$

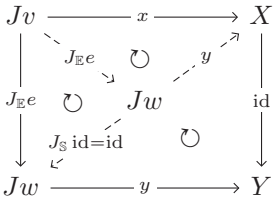
Again, bisimilarity can be defined as the existence of a span of open maps

Definition 4.3. *We say that X and Y are* ($\mathbb{E}, \mathbb{S}$)*-bisimilar if there is a span of* ($\mathbb{E}, \mathbb{S}$)*-open maps between them.*

4.2 Basic Properties

In this section, we will prove general properties of ($\mathbb{E}, \mathbb{S}$)-bisimilarity similar to the classical case. First, we show that if $\mathbb{M}$ has pullbacks, then ($\mathbb{E}, \mathbb{S}$)-bisimilarity is an equivalence relation. Secondly, we describe two notions of path bisimulations, both implied by ($\mathbb{E}, \mathbb{S}$)-bisimilarity. Finally, we prove that it is enough to check openness on some generators of $\mathbb{E}$.

In order to see when ($\mathbb{E}, \mathbb{S}$)-bisimilarity is an equivalence relation, we need to check symmetry, reflexivity, and transitivity. *Symmetry* always holds because we can always swap the legs of the span. For *reflexivity*, it is enough to prove that identities are open which is valid because $\mathbb{S}$ is a category and $J_{\mathbb{S}}$ is a functor, as shown in the diagram on the right. The proof of *transitivity* relies on composition and pullbacks:



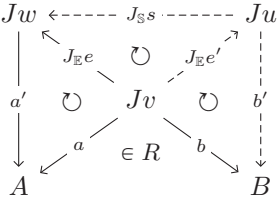
Lemma 4.4. ($\mathbb{E}, \mathbb{S}$)*-open maps are closed under composition and pullbacks.*

Theorem 4.5. *If $\mathbb{M}$ has pullbacks, then* ($\mathbb{E}, \mathbb{S}$)*-bisimilarity is a transitive relation, and thus is an equivalence relation.*

Generalized Path Bisimulations. In the classical open map setup [16], another notion of bisimilarity can be defined by using path extensions directly: so-called strong path and path bisimulations, which can be generalized as follows. Like originally [16], we assume that there is an element $0 \in V$, such that $J0$ is an initial object of $\mathbb{M}$ (note that 0 is not required to be initial in $\mathbb{E}$ or $\mathbb{S}$). The intuition is that the unique morphism $!_X \colon J0 \to X$ points to the initial state of X. For example, $J0$ can be given by $(1, \mathrm{id}_1, \bot)$ in a category of pointed coalgebras if 1 is the final object of $\mathcal{C}$ and if $\mathcal{C}(1, F1)$ has the least element $\bot \colon 1 \to F1$ (those conditions hold in the cases of interest).

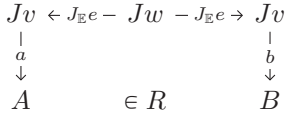
Definition 4.6. *A* path simulation *from A to B in $\mathbb{M}$ is a set R of spans of the form $A \xleftarrow{a} Jv \xrightarrow{b} B$ (for $v \in V$) satisfying the following two properties*

- ***initial condition:*** *the span $A \xleftarrow{!_A} J0 \xrightarrow{!_B} B$ belongs to R.*
- ***forward closure:*** *for all spans $A \xleftarrow{a} Jv \xrightarrow{b} B$ in R, all $e \colon v \to w \in \mathbb{E}$ and all $a' \colon Jw \to A \in \mathbb{M}$ such that $a = a' \cdot J_{\mathbb{E}}e$, there are $e' \colon v \to u \in \mathbb{E}$, $s \colon u \to w \in \mathbb{S}$, and $b' \colon Ju \to B \in \mathbb{M}$ such that $J_{\mathbb{E}}e = J_{\mathbb{S}}s \cdot J_{\mathbb{E}}e'$, $b = b' \cdot J_{\mathbb{E}}e'$, and the span $A \xleftarrow{a' \cdot J_{\mathbb{S}}s} Ju \xrightarrow{b'} B$ belongs to R.*



We say that R is a strong path simulation *if it additionally satisfies the following:*

- ***backward closure:*** *for all spans $A \xleftarrow{a} Jv \xrightarrow{b} B$ in R and all $e \colon w \to v \in \mathbb{E}$, we have that the span $A \xleftarrow{a \cdot J_{\mathbb{E}}e} Jw \xrightarrow{b \cdot J_{\mathbb{E}}e} B$ belongs to R.*



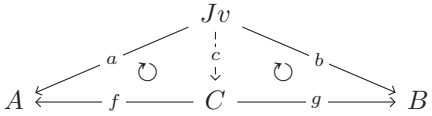
We say that R is a (strong) path bisimulation *from A to B if R and $R^\dagger = \{B \xleftarrow{b} Jv \xrightarrow{a} A \mid A \xleftarrow{a} Jv \xrightarrow{b} B \in R\}$ are (strong) path simulations.*

Remark that this version of (strong) path bisimulations has the same type as the one by Joyal et al. [16], but satisfies more general conditions. In particular, when $\mathbb{S}$ is a discrete category, the formulation above is exactly the one from [16]. Obviously, a strong path bisimulation is a path bisimulation.

The main result of this section is the following.

Theorem 4.7. *Assume two models A and B in $\mathbb{M}$. If there is a span $A \xleftarrow{f} C \xrightarrow{g} B$ where g is a morphism of $\mathbb{M}$ and f is an $(\mathbb{E}, \mathbb{S})$-open map, then the following set is a strong path simulation:*

$$R_{f,g} := \{A \xleftarrow{a} Jv \xrightarrow{b} B \mid \exists c \colon Jv \to C \text{ with } a = f \cdot c \text{ and } b = g \cdot c\}$$



Consequently, if A and B are $(\mathbb{E}, \mathbb{S})$-bisimilar, then there is strong path bisimulation between them.

As in the classical case of [16], there is no reason for the converse to be true in general: there might be a strong path bisimulation between two models, but no span of generalized open maps. However, conditions from [6] could be accommodated to describe a general framework in which the converse holds. Since this is not the main focus of this paper, we will not do it here, but will show a particular case in Section 6.

Generators of the Category of Extensions. In the first example of open maps for LTSs introduced in Section 2, the path category was described as the poset of words with the prefix order. Consequently, to prove that a functional simulation is J-open, we have to prove the lifting property of Definition 4.1 with respect to all pairs $w \leqslant w'$. However, it is sufficient to check the lifting property for extensions by one letter: $w' = w.a$ for some $a \in A$. The general reason is that, as a category, $(A^*, \leqslant)$ is generated by the morphisms $w \leqslant w.a$, and verifying the lifting property with respect to generators of the category $\mathbb{P}$ is enough to obtain J-openness. This can be extended to generalized open maps, with additional care.

Proposition 4.8. *Assume a subgraph $\mathbb{E}'$ of $\mathbb{E}$ that generates $\mathbb{E}$, that is, every morphism of $\mathbb{E}$ is a finite composition of morphisms of $\mathbb{E}'$. Assume additionally, that for every $e \in \mathbb{E}'$ and $s \in \mathbb{S}$ for which $J_{\mathbb{E}}e \cdot J_{\mathbb{S}}s$ is well-defined, there are $s' \in \mathbb{S}$ and $e' \in \mathbb{E}'$ such that $J_{\mathbb{E}}e \cdot J_{\mathbb{S}}s = J_{\mathbb{S}}s' \cdot J_{\mathbb{E}}e'$.*

In that case, if a morphism of $\mathbb{M}$ satisfies the lifting property of Definition 4.1 for all morphisms in $\mathbb{E}'$, then it is $(\mathbb{E}, \mathbb{S})$-open. Also, if a set of spans satisfies the conditions of Definition 4.6, where $\mathbb{E}$ is replaced by $\mathbb{E}'$, then it is a (strong) path bisimulation.

The first condition is satisfied when $\mathbb{E}$ is a free category and $\mathbb{E}'$ is its class of generators. The second condition is satisfied for e.g. $\mathbb{E} = \mathbb{P}$ and $\mathbb{S} = |\mathbb{P}|$.

5 Open Maps for Weighted Systems

In this section, we will prove that weighted systems can be captured by this generalized open map theory for a large variety of weights, including those needed to capture probabilistic systems.

5.1 Category of Coalgebras for Weighted Systems

In this section, we will consider weighted functors as follows.

Definition 5.1. *Given a commutative monoid $(K, +, e)$, the K-weighted functor $(K, +, e)^{(-)} : \mathsf{Set} \to \mathsf{Set}$ is defined as follows on sets and maps:*

$$\begin{array}{lrcl} \textit{sets:} & X & \mapsto & (K,+,e)^{(X)} = \{\mu : X \to K \mid \mu^{-1}(K \backslash \{e\}) \textit{ is finite}\} \\ \textit{maps:} & f : X \to Y & \mapsto & (K,+,e)^{(f)}(\mu) = \left(y \in Y \mapsto \sum \{\mu(x) \mid x \in X, f(x) = y\}\right) \end{array}$$

An element μ of $(K,+,e)^{(X)}$ is a finite distributions sending each $x \in X$ to a weight in K. Whenever a map $f\colon X \to Y$ identifies elements $f(x_1) = f(x_2) = \cdots$, then the functor action turns μ into a distribution on Y by adding up the weights $\mu(x_1) + \mu(x_2) + \cdots$ as elements of X are sent to the same element in Y. Since μ is finite and K is commutative, this addition is well-defined.

Given a commutative monoid $(K,+,e)$ and an alphabet A, we want to consider weighted systems as coalgebras for the functor $(K,+,e)^{(A\times -)}$. As described in Section 2, we want to be able to talk about lax homomorphisms, so we need an order on $(K,+,e)^{(A\times -)}$ as in Definition 2.8. For that, we need to assume an ordered commutative monoid $(K,+,e,\sqsubseteq)$, that is, a monoid $(K,+,e)$ with a partial order $\sqsubseteq$ such that $+$ is monotone in both its arguments.

Lemma 5.2. *Given an ordered commutative monoid $(K,+,e,\sqsubseteq)$, then for all sets X and Y, the relation on the hom-set $\mathsf{Set}\big(X,(K,+,e)^{(A\times Y)}\big)$ defined by*

$$f_1 \sqsubseteq f_2 \iff \forall x \in X,\ \forall y \in Y,\ \forall a \in A,\ f_1(x)(a,y) \sqsubseteq f_2(x)(a,y)$$

is an order on $(K,+,e)^{(A\times -)}$.

So, we have a category $\mathsf{LCoalg}\left(1,(K,+,e)^{(A\times -)}\right)$ of pointed coalgebras and lax homomorphisms. The goal of this section is to design a generalized open maps situation for which $(\mathbb{E},\mathbb{S})$-bisimilarity characterizes coalgebraic bisimilarity and more precisely for which $(\mathbb{E},\mathbb{S})$-openness characterizes proper homomorphisms.

In the course of the constructions and proofs, we will need additional assumptions that we list here.

Definition 5.3. *We call an ordered commutative monoid $(K,+,e,\sqsubseteq)$ a* rearrangement monoid *if it satisfies the additional requirement that if $n,m \geqslant 1$ and*

$$\sum_{i=1}^{n} x_i \sqsubseteq \sum_{j=1}^{m} y_j,$$

then there exists a family $(u_{i,j})_{1\leqslant i\leqslant n,1\leqslant j\leqslant m}$ such that

$$\text{for all } j,\ \sum_{i=1}^{n} u_{i,j} \sqsubseteq y_j \quad \text{and} \quad \text{for all } i,\ \sum_{j=1}^{m} u_{i,j} = x_i.$$

In addition, we say that a rearrangement monoid is strict *if the condition above holds also when replacing $\sqsubseteq$ with $=$.*

The intuition is as follows. We have some weights arranged as $x_1,\dots,x_n$. We want to be able to decompose those weights into smaller weights, the $u_{i,j}$s, and by rearranging those small weights obtaining weights smaller than the y_j. This condition states that this is possible when there is enough weight in total. The special case of strictness is called the *row-column property* in [17].

Lemma 5.4. *For any subgroup G of the real numbers $(\mathbb{R}^n, +, -, 0)$ such that for all x, y in G $(\min(x_1, y_1), \ldots, \min(x_n, y_n)) \in G$, the monoids $(G, +, 0, \leqslant)$ and $(G_{\geqslant 0}, +, 0, \leqslant)$, where $\leqslant$ is the usual order on $\mathbb{R}^n$, are strict rearrangement monoids.*

For any lattice with bottom element $(L, \leqslant, \sqcup, \sqcap, \bot)$, $(L, \sqcup, \bot, \leqslant)$ is a rearrangement monoid if and only if $(L, \leqslant, \sqcup, \sqcap)$ is distributive. Furthermore, in that case, it is always strict.

Another property is a form of positivity: we say that an ordered monoid is *positively ordered* if e is the bottom element for $\sqsubseteq$, that is, for all $k \in K$, $e \sqsubseteq k$.

Example 5.5. The positive real line $(\mathbb{R}_+, +, 0, \leqslant)$ is a positively ordered strict rearrangement monoid and it is necessary to define probabilistic systems. Another example is the monoid of natural numbers $(\mathbb{N}, +, 0, \leqslant)$, which defines the bag functor. Finally, any distributive lattice with bottom element $(L, \sqcup, \bot, \leqslant)$, typically powerset lattices $(\mathcal{P}(X), \cup, \varnothing, \subseteq)$, is too. On the contrary, $(\mathbb{R}, +, 0, \leqslant)$ and $(\mathbb{Z}, +, 0, \leqslant)$ are strict rearrangement monoids but are not positively ordered. Conversely $(\mathbb{N}_{\geqslant 1}, \times, 1, \leqslant)$ is positively ordered but not a rearrangement monoid. Indeed, it is impossible to rearrange the inequality $2 \times 5 \leqslant 3 \times 4$.

5.2 Generalized Open Maps Situation for Weighted Systems

Let $(K, +, e, \sqsubseteq)$ be a commutative ordered monoid. Elements of V_K are

- either words on $A \times (K \backslash \{e\})$, $w = (a_1, k_1), \ldots, (a_n, k_n)$,
- or triples (w_1, b, w_2) of a word w_1 on $A \times (K \backslash \{e\})$, a letter $b \in A$, and a non-empty word w_2 on $(K \backslash \{e\})$.

The function J_K maps

- a word $w = (a_1, k_1), \ldots, (a_n, k_n)$ to the system

$$Jw = \boxed{0 \xrightarrow{(a_1,k_1)} 1 \xrightarrow{(a_2,k_2)} \cdots \xrightarrow{(a_n,k_n)} n}$$

  that is, to the coalgebra $Jw \colon \{0, \ldots, n\} \to K^{(A \times \{0,\ldots,n\})}$ such that if $b = a_{i+1}$ and $j = i + 1$ then $Jw(i)(b, j) = k_{i+1}$, else $= e$.
- a triple (w_1, b, w_2) with $w_1 = (a_1, k_1), \ldots, (a_n, k_n)$ and $w_2 = l_1, \ldots, l_m$ is mapped to the system

$$J(w_1, b, w_2) = \boxed{0 \xrightarrow{(a_1,k_1)} 1 \xrightarrow{(a_2,k_2)} \cdots \xrightarrow{(a_n,k_n)} n \begin{array}{l} \xrightarrow{(b,l_1)} (n+1, 1) \\ \vdots \\ \xrightarrow{(b,l_m)} (n+1, m) \end{array}}$$

  that is, $J(w_1, b, w_2)(n)(n + 1, i) = (b, l_i)$.

The category $\mathbb{E}_K$ is defined as follows. For every w_1, b, and w_2, there is a unique edge e from w_1 to (w_1, b, w_2). The functor then maps this edge e to $J_{\mathbb{E}} e$, the obvious injection.

The category $\mathbb{S}_K$ has two types of morphisms:

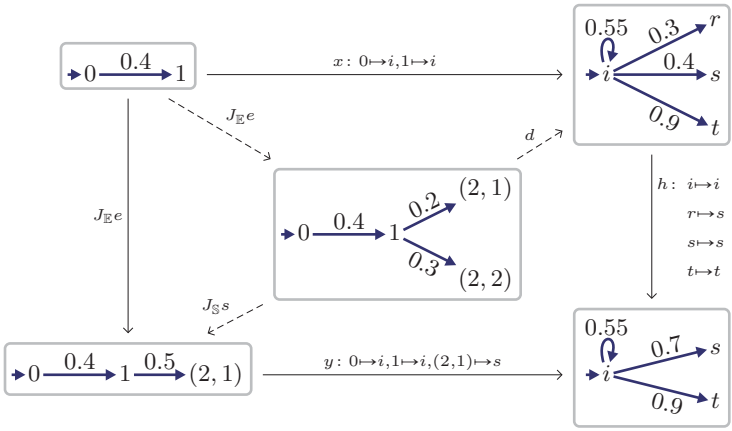


Fig. 1. Example of a lifting of a path extension in $\mathbb{R}_+$-weighted systems and for a singleton label alphabet $|A| = 1$, thus omitting action labels.

- identities on words w_1,
- morphisms from (w_1, b, w_2') to (w_1, b, w_2), with $w_2' = l_1', \ldots, l_{m'}'$ and $m \leqslant m'$, which are given by surjective monotone functions $s\colon \{1, \ldots, m'\} \to \{1, \ldots, m\}$ such that for all $i \leqslant m$, $l_i = \sum_{\{j \mid s(j)=i\}} l_j'$.

The functor $J_{\mathbb{S}}$ then maps s of the second type to the proper homomorphism $J_{\mathbb{S}}s$ which maps i to i and $(n+1, j)$ to $(n+1, s(j))$.

As a piece of notation, for a morphism $x\colon Jw_1 \to X$, with w_1 of length n we denote $x(n) \in X$ by $\mathrm{end}(x)$. We then say that a state p of X is reachable if there is a morphism of type $x\colon Jw_1 \to X$ with $\mathrm{end}(x) = p$. By extension, we say that X is reachable if all its states are reachable.

5.3 Equivalence between Open Maps and Proper Homomorphisms

An example of an $(\mathbb{E}, \mathbb{S})$-open map h is provided in Figure 1, together with a path extension that is lifted. Like it is often the case in the non-deterministic systems, the lifting map d is not unique. Hence, only existence (and no uniqueness) is required in the lifting property. Since h is a proper homomorphism, it provides a lifting for all extensions, as we show in general:

Theorem 5.6. *Assume a lax homomorphism $f\colon X \to Y$. If f is $(\mathbb{E}_K, \mathbb{S}_K)$-open, X is reachable, and K is positively ordered, then f is a proper homomorphism. Conversely, if f is a proper homomorphism and K is a rearrangement monoid, then f is $(\mathbb{E}_K, \mathbb{S}_K)$-open. In particular, if K is a positively ordered rearrangement monoid, two weighted systems X and Y are $(\mathbb{E}_K, \mathbb{S}_K)$-bisimilar if and only if they are coalgebraically bisimilar.*

For an endofunctor on Set, to prove that coalgebraic bisimilarity is an equivalence relation it is enough to show that the functor preserves weak-pullbacks. In the case of the weighted functor, this is given by strictness (see also [17]):

Corollary 5.7. *If K is a positively ordered strict rearrangement monoid, then $(\mathbb{E}_K, \mathbb{S}_K)$-bisimilarity is an equivalence relation.*

5.4 About Sub-distribution Functor

Until now, we have not dealt with probabilistic systems, that is, coalgebras for the sub-distribution functor $\mathcal{D}_{\leqslant 1}$. Those coalgebras are particular cases of coalgebras for the weighted functor $X \mapsto (\mathbb{R}_+, +)^{(X)}$. We want to show in this section that it is equivalent to consider coalgebras for $X \mapsto \mathcal{D}_{\leqslant 1}(A \times X)$ as coalgebras for $X \mapsto (\mathbb{R}_+, +)^{(A \times X)}$, in the sense that, two coalgebras for the former are bisimilar if and only if they are bisimilar when seen as coalgebras for the latter. The main ingredient is the following remark.

Lemma 5.8. *Assume a pointed coalgebra $1 \xrightarrow{i} X \xrightarrow{c} \mathcal{D}_{\leqslant 1}(A \times X)$ and assume given a lax (resp. proper) homomorphism f from $1 \xrightarrow{j} Y \xrightarrow{d} (\mathbb{R}_+, +)^{(A \times Y)}$ to $1 \xrightarrow{i} X \xrightarrow{c} \mathcal{D}_{\leqslant 1}(A \times X) \subseteq (\mathbb{R}_+, +)^{(A \times X)}$. Then $Y \xrightarrow{d} \mathcal{D}_{\leqslant 1}(A \times Y)$ and f is a lax (resp. proper) homomorphism from $1 \xrightarrow{j} Y \xrightarrow{d} \mathcal{D}_{\leqslant 1}(A \times Y)$ to $1 \xrightarrow{i} X \xrightarrow{c} \mathcal{D}_{\leqslant 1}(A \times X)$.*

Remark that this property is not true for the proper distribution functor $\mathcal{D}$. This suggests that we can define a generalized open maps situation $\mathbb{E}_{\mathcal{D}}, \mathbb{S}_{\mathcal{D}}$ for coalgebras for the functor $X \mapsto \mathcal{D}_{\leqslant 1}(A \times X)$ by considering $\mathbb{E}_{(\mathbb{R}_+,+)}, \mathbb{S}_{(\mathbb{R}_+,+)}$ as defined in Section 5.2, and restricting it to those v such that Jv is a coalgebra for $X \mapsto \mathcal{D}_{\leqslant 1}(A \times X)$.

Corollary 5.9. *A lax homomorphism from $1 \xrightarrow{j} Y \xrightarrow{d} \mathcal{D}_{\leqslant 1}(A \times Y)$ to $1 \xrightarrow{i} X \xrightarrow{c} \mathcal{D}_{\leqslant 1}(A \times X)$ is $(\mathbb{E}_{\mathcal{D}}, \mathbb{S}_{\mathcal{D}})$-open if and only if it is $(\mathbb{E}_{(\mathbb{R}_+,+)}, \mathbb{S}_{(\mathbb{R}_+,+)})$-open. Furthermore, two $\mathcal{D}_{\leqslant 1}(A \times \_)$-coalgebras are $(\mathbb{E}_{\mathcal{D}}, \mathbb{S}_{\mathcal{D}})$-bisimilar if and only if they are $(\mathbb{E}_{(\mathbb{R}_+,+)}, \mathbb{S}_{(\mathbb{R}_+,+)})$-bisimilar.*

Finally, the main result of this section:

Theorem 5.10. *Let $f \colon X \to Y$ be a lax homomorphism between $\mathcal{D}_{\leqslant 1}(A \times \_\_)$-coalgebras (X, c, i) and (Y, d, j). If (X, c, i) is reachable and f is $(\mathbb{E}_{\mathcal{D}}, \mathbb{S}_{\mathcal{D}})$-open, then f is a proper homomorphism. Conversely, if f is a proper homomorphism, then it is $(\mathbb{E}_{\mathcal{D}}, \mathbb{S}_{\mathcal{D}})$-open. Moreover, two $\mathcal{D}_{\leqslant 1}(A \times \_\_)$-coalgebras (X, c, i) and (Y, d, j) are $(\mathbb{E}_{\mathcal{D}}, \mathbb{S}_{\mathcal{D}})$-bisimilar if and only if they are coalgebraically bisimilar.*

6 Open Maps for Branching Bisimilarity

In this section, we present a new way of modeling branching and weak bisimulations using our generalized framework of open maps. Using this additional flexibility, we do not need to rely on weak morphisms anymore, but on a slight modification of the morphism described in Definition 2.1. Concretely, we build a

generalized open map situation such that stuttering branching bisimulations coincide with strong path bisimulations, and that in this case, they precisely characterize $(\mathbb{E}, \mathbb{S})$-bisimilarity. In addition, in this framework, path bisimulations precisely correspond to weak bisimulations, witnessing branching bisimilarity as the history-preserving analogue to weak bisimilarity.

6.1 LTSs with Internal Moves, Category and Bisimilarities

Definition 6.1. *For a fixed set A of labels with a particular element τ (called* internal move*), the category* WLTS *contains the same objects as* LTS*, and its morphisms $f\colon (X, \to, x_0) \to (Y, \to, y_0)$ are functions $f\colon X \to Y$ such that $f(x_0) = y_0$ and for all $x \xrightarrow{a} x'$ in X, we have $f(x) \xrightarrow{a} f(x')$, or $a = \tau$ and $f(x) = f(x')$.*

LTS is a (non-full) subcategory of WLTS, and in fact the LTS-morphisms will be used later in the paper. For easier distinction, we use the terminology *strong morphisms* for WLTS-morphisms that are also in LTS (alluding to *strong bisimulations* which were the bisimulation notion in LTS). Another notion of morphisms are so-called *weak morphisms* [3]:

- if $x \xrightarrow{a} x'$ in X, then $f(x) \xrightarrow{\tau}{}^{\star} \xrightarrow{a} \xrightarrow{\tau}{}^{\star} f(x')$ in Y,
- if $x \xrightarrow{\tau} x'$ in X, then $f(x) \xrightarrow{\tau}{}^{\star} f(x')$ in Y.

Though we do not use weak morphisms in the following development of the paper, it is worth mentioning the WLTS-morphisms form a proper subclass of the weak morphisms.

Definition 6.2. *A* branching bisimulation *from $(X, \to_X, i_X)$ to $(Y, \to_Y, i_Y)$ is a relation $R \subseteq X \times Y$ such that $(i_X, i_Y) \in R$, and for $(x, y) \in R$:*

- *if $x \xrightarrow{a} x'$ then*
  - *$a = \tau$ and $(x', y) \in R$, or*
  - *$y \xrightarrow{\tau} y_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} y_n \xrightarrow{a} z_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} z_m$ such that (x, y_n), (x', z_1), and $(x', z_m) \in R$.*
- *symmetrically when $y \xrightarrow{a} y'$.*

If furthermore in the second condition (x, y_i), $(x', z_i) \in R$ for all i (and symmetrically in the third condition), then R is said to be stuttering.

It is known from [23] that the largest branching bisimulation is stuttering, so that both notions generate the same bisimilarity. In the following, we will prove that strong path bisimulations are more naturally related to stuttering branching bisimulations thanks to their backward closure.

Definition 6.3. *A* weak bisimulation *from $(X, \to_X, i_X)$ to $(Y, \to_Y, i_Y)$ is a relation $R \subseteq X \times Y$ such that $(i_X, i_Y) \in R$, and for $(x, y) \in R$:*

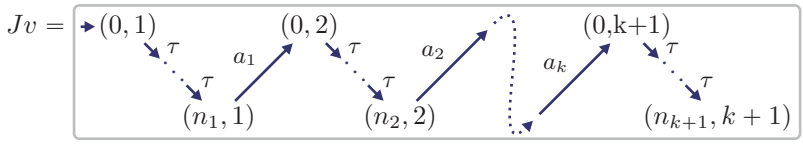
- *if $x \xrightarrow{\tau} x'$, then there is y' such that $(x', y') \in R$ and $y \xrightarrow{\tau}{}^{\star} y'$,*
- *if $x \xrightarrow{a} x'$ with $a \neq \tau$, then there is y' such that $(x', y') \in R$ and $y \xrightarrow{\tau}{}^{\star} \xrightarrow{a} \xrightarrow{\tau}{}^{\star} y'$.*
- *symmetrically when $y \xrightarrow{\tau} y'$ or $y \xrightarrow{a} y'$.*

It is clear that a (stuttering) branching bisimulation is a weak bisimulation.

6.2 Generalized Open Maps for Branching Bisimulations

In this section, we describe the generalized open maps situation that captures branching bisimulation. Like for plain LTSs (Def. 2.2), elements of V will be words on A, representing a finite linear LTS labelled by this word. However, to emphasize the particularity of the internal move τ, we will provide another presentation here.

Here, V is the set of sequences of the form: $v = n_1, a_1, n_2, \ldots, n_k, a_k, n_{k+1}$ such that $a_i \in A \backslash \{\tau\}$ and $n_i \in \mathbb{N}$, e.g. $\tau\tau a \tau bc\tau \mathrel{\widehat{=}} 2, a, 1, b, 0, c, 1$. The natural numbers $n_i \in \mathbb{N} \cong \{\tau\}^*$ represent the number of internal moves between two observable moves. Then, J maps this sequence to the usual linear LTS:



Elements of $\mathbb{E}$ append *at most* one observable (i.e. non-τ) move:

- **Only internal moves**: for sequences $v = n_1, a_1, \ldots, a_k, n_{k+1}$ and $w = n_1, a_1, \ldots, a_k, n'_{k+1}$ with $n_{k+1} \leqslant n'_{k+1}$ there is a unique edge $e_\tau \colon v \to w$ in $\mathbb{E}$, e.g. $e_\tau \colon 2, a, 1, b, 0, c, 1 \to 2, a, 1, b, 0, c, 3$
- **One observable move**: for sequences $v = n_1, a_1, \ldots, a_k, n_{k+1}$ and $w = n_1, a_1, \ldots, a_k, n'_{k+1}, a, n_{k+2}$ with $n_{k+1} \leqslant n'_{k+1}$ there is a unique edge $e_a \colon v \to w$ in $\mathbb{E}$.

The graph morphism $J_{\mathbb{E}} \colon \mathbb{E} \to \mathbb{M}$ maps those edges to the obvious inclusion, mapping state (i, j) of Jv to the same state in Jw.

Strictly speaking, $\mathbb{E}$ is not a category, but just a graph, because we have $a \xrightarrow{e_b} ab$ and $ab \xrightarrow{e_c} abc$, but there is no morphism from a to abc. To fit in the framework of Section 4, we take the free category Free($\mathbb{E}$) generated by this graph and the unique functor extending the graph homomorphism $J_{\mathbb{E}}$. By Proposition 4.8, it is equivalent to consider Free($\mathbb{E}$) and $\mathbb{E}$ for openness and path bisimulations, so we will talk of $(\mathbb{E}, \mathbb{S})$-openness, when we mean $(\text{Free}(\mathbb{E}), \mathbb{S})$-openness, and all the statements and proofs will be done using $\mathbb{E}$ only.

Elements of $\mathbb{S}$ are trickier to describe. The intuition is that they are morphisms that merge states. In the context of LTSs with internal moves, merging happens when the source and the target of a τ-transition are mapped to the same state. This is crucial for the open maps we want to describe: to lift one τ-transition, it might be necessary to use several τ-transitions. With this knowledge, elements of $\mathbb{S}$ are as follows.

- **Merging internal moves**: morphisms in $\mathbb{S}$ from $v = n_1, a_1, \ldots, a_k, n_{k+1}$ to $w = n'_1, a_1, \ldots, a_k, n'_{k+1}$ with $n_i \geqslant n'_i$ are $(k+1)$-tuples $s = (s_1, \ldots, s_{k+1})$ of monotone surjective functions $s_i \colon \{0 < 1 < \ldots < n_i\} \to \{0 < 1 < \ldots < n'_i\}$.

For example, there are two morphisms from $a\tau\tau b \mathrel{\widehat{=}} 0, a, 2, b, 0$ to $a\tau b \mathrel{\widehat{=}} 0, a, 1, b, 0$, one for each τ that can be dropped. The functor $J_{\mathbb{S}}$ then maps s to the morphism from Jv to Jw defined by $J_{\mathbb{S}}(s)(i, j) = (s_j(i), j)$.

As a piece of notation, for a morphism $x\colon J(n_1, a_1, \dots, a_k, n_{k+1}) \to X$, we denote $x(n_{k+1}, k+1) \in X$ by $\mathrm{end}(x)$.

6.3 Equivalence of Bisimilarities

In this section, we prove that $(\mathbb{E}, \mathbb{S})$-bisimilarity indeed coincides with branching bisimilarity. To do so, we prove first that for the present instance of $\mathbb{E}$ and $\mathbb{S}$ (Sec. 6.2), $(\mathbb{E}, \mathbb{S})$-bisimilarity coincides with strong path bisimilarity. In general, $(\mathbb{E}, \mathbb{S})$-bisimilarity implies strong path bisimilarity (Theorem 4.7), so it remains to show the converse direction for the present instance. To this end, we start by internalizing strong path bisimulations into objects of $\mathsf{LTS}/\mathsf{WLTS}$, in order to relate it them to open maps:

Definition 6.4. *For a strong path bisimulation R from X to Y, define the LTS $\widetilde{R} = (R, \to_R, (X \xleftarrow{!} J0 \xrightarrow{!} Y))$ to have transitions*

$$(X \xleftarrow{x} Jv \xrightarrow{y} Y) \xrightarrow{a}_R (X \xleftarrow{x'} Jw \xrightarrow{y'} Y)$$

- *for $a \neq \tau$ with $v = (n_1, a_1, \dots, a_k, n_{k+1})$, $w = (n_0, a_1, \dots, a_k, n_{k+1}, a, 0)$, $x' = x \cdot J_{\mathbb{E}} e_a$, and $y' = y \cdot J_{\mathbb{E}} e_a$ (for the unique $e_a\colon v \to w$);*
- *for $a = \tau$ with $v = (n_1, a_1, \dots, a_k, n_{k+1})$, $w = (n_1, a_1, \dots, a_k, n_{k+1} + 1)$, $x' = x \cdot J_{\mathbb{E}} e_\tau$, and $y' = y \cdot J_{\mathbb{E}} e_\tau$ (for the unique $e_\tau\colon v \to w$).*

As a first observation, we describe runs in $\widetilde{R}$ in terms of projection maps:

Lemma 6.5. *In WLTS, we have projection maps $X \xleftarrow{\pi_X} \widetilde{R} \xrightarrow{\pi_Y} Y$ given by $\pi_X\colon (X \xleftarrow{x} Jv \xrightarrow{y} Y) \mapsto \mathrm{end}(x)$ and $\pi_Y\colon (X \xleftarrow{x} Jv \xrightarrow{y} Y) \mapsto \mathrm{end}(y)$. For every strong morphism $r\colon Jv \to \widetilde{R}$ (i.e. $r \in \mathsf{LTS}$),*

$$\mathrm{end}(r) \text{ is of the form } (X \xleftarrow{\pi_X \cdot r} Jv \xrightarrow{\pi_Y \cdot r} Y).$$

Remark that in this statement, we require r to be strong and not just a morphism of WLTS. With a morphism of WLTS, the statement would become that there is $s\colon v' \to v \in \mathbb{S}$ such that $\pi_X \cdot r = x \cdot J_{\mathbb{S}} s$ instead. For the characterization of open maps in WLTS, it suffices for our needs to restrict to strong morphisms:

Lemma 6.6. *For $f\colon X \to Y$ in WLTS to be $(\mathbb{E}, \mathbb{S})$-open, it is sufficient to verify the lifting in Definition 4.1 in the special case of x being a strong morphism.*

We use this simplification to prove that the projection maps π_X, π_Y are open:

Proposition 6.7. *For a strong path bisimulation R from X to Y, the projections $X \xleftarrow{\pi_X} \widetilde{R} \xrightarrow{\pi_Y} Y$ are both $(\mathbb{E}, \mathbb{S})$-open.*

The next step is to prove the equivalence between strong path and stuttering branching bisimulations.

Table 1. Equivalences of bisimilarity notions in LTSs with τ-actions $X, Y \in \mathsf{WLTS}$

branching bisimilarity	$\iff$ strong path bisimilarity	(Theorem 6.8)
	$\iff$ $(\mathbb{E}, \mathbb{S})$-bisimilarity	(Proposition 6.7 & Theorem 4.7)
weak bisimilarity	$\iff$ path bisimilarity	(Theorem 6.9)

Theorem 6.8. *If R is a stuttering branching bisimulation from X to Y, then*

$$\overline{R} = \{X \xleftarrow{x} Jv \xrightarrow{y} Y \mid v = (n_1, a_1, \ldots, n_{k+1}) \wedge \forall i, j.\, (x(i,j), y(i,j)) \in R\}$$

is a strong path bisimulation. Conversely, if R is a strong path bisimulation, then

$$\breve{R} = \{(end(x), end(y)) \mid (X \xleftarrow{x} Jv \xrightarrow{y} Y) \in R\}$$

is a stuttering branching bisimulation.

The same reasoning can be made for weak and path bisimulations:

Theorem 6.9. *If R is a weak bisimulation from X to Y, then*

$$\widehat{R} = \{X \xleftarrow{x} Jv \xrightarrow{y} Y \mid (end(x), end(y)) \in R\}$$

is a path bisimulation. If R is a path bisimulation, then $\breve{R}$ is a weak bisimulation.

In total, we can describe branching and weak bisimilarity by categorical bisimilarity notions, as summarized in Table 1.

7 Conclusions and Future Work

In this paper, we investigate bisimilarities of weighted and probabilistic systems through the theory of open maps. After showing that the usual theory cannot capture weights, we provide a faithful extension of the theory by the notion of mergings. The new theory has similar properties (equivalence relation, characterization as sets of spans, restriction to generators) as classical open maps but also captures bisimilarity of weighted systems and even branching bisimilarity.

The new instances come at the cost of more parameters to the theory. It remains for future work whether the parameters $\mathbb{E}$, $\mathbb{S}$ can be combined in a single path category with two morphism classes and morphism factorizations. It would also be illuminating to know whether this new theory satisfies the axioms of a *class of open maps* from [15], in particular for toposes of coalgebras [14].

For the framework as presented, we would like to formally relate it to coalgebra – as this has been done for non-deterministic systems [19,25]. Furthermore, we would like to investigate how system semantics of true concurrency, such as Higher Dimensional Automata [21] can be integrated. Designing open maps for them turned out to be complicated (see [8]), but a hope would be that the addition of mergings would allow modeling homotopy more naturally.

Finally, it would be interesting to see whether our theory capture quantitative extensions of systems classically modeled by open maps, such as probabilistic and quantum extensions of petri nets and event structures (see [24] for example).

References

1. Beohar, H., Cuijpers, P.J.L.: Open Maps in Concrete Categories and Branching Bisimulation for Prefix Orders. Electronic Notes in Theoretical Computer Science **319**, 51–66 (2015). https://doi.org/10.1016/j.entcs.2015.12.005
2. Beohar, H., Küpper, S.: Bisimulation Maps in Presheaf Categories. Electronic Notes in Theoretical Computer Science **347**, 5–24 (2019). https://doi.org/10.1016/j.entcs.2019.09.002
3. Cheng, A., Nielsen, M.: Open Maps (at) Work. B R I C S Report Series (RS-95-23) (1995)
4. Danos, V., Desharnais, J., Laviolette, F., Panangaden, P.: Bisimulation and cocongruence for probabilistic systems. Information and Computation **204**(4), 503–523 (2006). https://doi.org/10.1016/j.ic.2005.02.004
5. Desharnais, J., Edalat, A., Panangaden, P.: Bisimulation for Labelled Markov Processes. Information and Computation **179**(2), 163–193 (2003). https://doi.org/10.1006/inco.2001.2962
6. Dubut, J., Goubault, E., Goubault-Larrecq, J.: Bisimulations and unfolding in P-accessible categorical models. In: Desharnais, J., Jagadeesan, R. (eds.) 27th International Conference on Concurrency Theory, CONCUR 2016. LIPIcs, vol. 59, pp. 25:1–25:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.CONCUR.2016.25
7. Dubut, J., Hasuo, I., Katsumata, S., Sprunger, D.: Quantitative bisimulations using coreflections and open morphisms (2018), arXiv:1809.09278
8. Fahrenberg, U., Legay, A.: History-Preserving Bisimilarity for Higher-Dimensional Automata via Open Maps. Electronic Notes in Theoretical Computer Science **298**, 165–178 (2013). https://doi.org/10.1016/j.entcs.2013.09.012
9. Fiore, M., Cattani, G.L., Winskel, G.: Weak bisimulation and open maps. In: Proceedings. 14th Symposium on Logic in Computer Science. pp. 67–76 (1999). https://doi.org/10.1109/LICS.1999.782590
10. Freyd, P., Scedrov, A.: Categories, Allegories, Mathematical Library, vol. 39. North-Holland (1990)
11. Hughes, J., Jacobs, B.: Simulations in coalgebra. Theor. Comput. Sci. **327**(1-2), 71–108 (2004). https://doi.org/10.1016/j.tcs.2004.07.022
12. Hune, T., Nielsen, M.: Timed bisimulations and open maps. In: Brim, L., Gruska, J., Zlatuška, J. (eds) Mathematical Foundations of Computer Science 1998. MFCS 1998. Lecture Notes in Computer Science, vol. 1450. Springer, Berlin, Heidelberg (1998). https://doi.org/10.1007/BFb0055787
13. Jacobs, B.: Introduction to Coalgebra: Towards Mathematics of States and Observation, Cambridge Tracts in Theoretical Computer Science, vol. 59. Cambridge University Press (2016)
14. Johnstone, P., Power, J., Tsujishita, T., Watanabe, H., Worrell, J.: On the structure of categories of coalgebras. Theoretical Computer Science **260**, 87–117 (2001). https://doi.org/10.1016/S0304-3975(00)00124-9
15. Joyal, A., Moerdijk, I.: A completeness theorem for open maps. Annals of Pure and Applied Logic **70**, 51–86 (1994). https://doi.org/10.1016/0168-0072(94)90069-8
16. Joyal, A., Nielsen, M., Winskel, G.: Bisimulation from Open Maps. Information and Computation **127**, 164–185 (1996). https://doi.org/10.1006/inco.1996.0057
17. Klin, B.: Semantics and Algebraic Specification, Lecture Notes in Computer Science, vol. 5700, chap. Structural Operational Semantics for Weighted Transition Systems, pp. 121–139. Springer, Berlin, Heidelberg (2009)

18. Larsen, K.G., Skou, A.: Bisimulations through Probabilistic Testing. Information and Computation **94**, 1–28 (1991). https://doi.org/10.1016/0890-5401(91)90030-6
19. Lasota, S.: Coalgebra morphisms subsume open maps. Theoretical Computer Science **280**(1), 123 – 135 (2002). https://doi.org/10.1016/S0304-3975(01)00023-8
20. Park, D.: Concurrency and automata on infinite sequences. In: Proceedings of the 5th GI-Conference on Theoretical Computer Science. Lecture Notes in Computer Science, vol. 104, pp. 167–183. Springer (1981). https://doi.org/10.1007/BFb0017309
21. Pratt, V.: Higher dimensional automata revisited. Mathenatical Structures in Computer Science **10**(4), 525–548 (2000). https://doi.org/10.1017/S0960129500003169
22. Rutten, J.: Universal coalgebra: a theory of systems. Theoretical Computer Science **249**(1), 3 – 80 (2000). https://doi.org/10.1016/S0304-3975(00)00056-6
23. van Glabbeek, R.J., Weijland, W.P.: Branching Time and Abstraction in Bisimulation Semantics. Journal of the ACM **43**(3), 555–600 (1996). https://doi.org/10.1145/233551.233556
24. Winskel, G.: Distributed probabilistic and quantum strategies. Electronic Notes in Theoretical Computer Science **298**, 403–425 (2013). https://doi.org/10.1016/j.entcs.2013.09.024
25. Wißmann, T., Dubut, J., Katsumata, S., Hasuo, I.: Path category for free. In: Bojańczyk, M., Simpson, A. (eds.) Foundations of Software Science and Computation Structures (FoSSaCS 2019). pp. 523–540. Springer International Publishing, Cham (04 2019). https://doi.org/10.1007/978-3-030-17127-8_30

Preservation and Reflection of Bisimilarity via Invertible Steps

Ruben Turkenburg(✉)[1], Clemens Kupke[2], Jurriaan Rot[1], and Ezra Schoen[2]

[1] Institute for Computing and Information Sciences (iCIS), Radboud University, Nijmegen, The Netherlands
ruben.turkenburg@ru.nl

[2] Department of Computer and Information Sciences, Strathclyde University, Glasgow, UK

Abstract. In the theory of coalgebras, distributive laws give a general perspective on determinisation and other automata constructions. This perspective has recently been extended to include so-called weak distributive laws, covering several constructions on state-based systems that are not captured by regular distributive laws, such as the construction of a belief-state transformer from a probabilistic automaton, and ultrafilter extensions of Kripke frames.
In this paper we first observe that weak distributive laws give rise to the more general notion of what we call an invertible step: a pair of natural transformations that allows to move coalgebras along an adjunction. Our main result is that part of the construction induced by an invertible step preserves and reflects bisimilarity. This covers results that have previously been shown by hand for the instances of ultrafilter extensions and belief-state transformers.

Keywords: Coalgebra · Bisimulations · Weak distributive laws

1 Introduction

Distributive laws between a monad T and a functor B are ubiquitous in the theory of coalgebras. They capture various forms of interaction between algebras and coalgebras, including structural operational semantics [45,33], efficient proof techniques [9] and a general coalgebraic determinisation procedure which applies to a wide range of automata and other state-based systems [43,15,29].

The central idea of this general determinisation procedure is to interpret coalgebras in the Eilenberg-Moore category $\mathcal{EM}(T)$, as coalgebras for a lifting of B that arises from the distributive law. Behavioural equivalence in $\mathcal{EM}(T)$ then amounts to desired notions of equivalence. For instance: language equivalence of non-deterministic automata; weighted automata [7]; Mealy and Moore machines with side-effects [43]; or various types of trace equivalence of transition systems [8].

An illustrative *non-example* of this general determinisation procedure is in a natural construction of belief-state transformers from probabilistic automata, which feature both non-determinism and probabilities. From a categorical perspective, the problem is related to the classical result that there is no suitable distributive law of the probability distribution monad $\mathcal{D}$ over the powerset monad

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 328–348, 2023.
https://doi.org/10.1007/978-3-031-30829-1_16

$\mathcal{P}$ [46] (also see [47,34] for other non-existence results of distributive laws). Hence, general determinisation via distributive laws seems not applicable here.

Nevertheless, in [12] a concrete coalgebraic account of the construction of belief-state transformers is given, in terms of a two-stage process:

1. from probabilistic automata to coalgebras in $\mathcal{EM}(\mathcal{D})$, which are a type of labelled transition systems over convex algebras;
2. from these coalgebras in $\mathcal{EM}(\mathcal{D})$ back to plain transition systems in Set, yielding the belief-state transformer.

A key result in *op. cit.* is that the second stage preserves and reflects behavioural equivalence. This shows that behavioural equivalence of coalgebras in $\mathcal{EM}(\mathcal{D})$ coincides with distribution bisimilarity on the belief-state transformer.

In [12,21] it was shown that this construction, in fact, arises from a canonical *weak distributive law* of $\mathcal{D}$ over $\mathcal{P}$ [22]. Weak distributive laws correspond to so-called weak liftings [19], and—as shown in [22]—these yield a new generalised determinisation procedure which covers the above example, and precisely instantiates to the two stages above. Further examples are the treatment of alternating automata via weak distributive laws in [23], and weak distributive laws for combining non-determinism with semimodules in [10].

However, the result for probabilistic automata that the second stage above preserves and reflects behavioural equivalence has not yet been accounted for in the abstract theory of determinisation via weak distributive laws.

In this paper we provide such an account, starting from a more general setting than weak distributive laws: what we call *invertible steps*. These basically replace the Eilenberg-Moore adjunction inherent in the weak liftings approach by a general adjunction. In this context, a *step* allows one to lift the left adjoint to coalgebras—this is a widely occurring phenomenon, for instance in the semantics of coalgebraic modal logic, testing semantics and trace semantics (see [41] for an overview). The key idea here is to assume a right inverse, allowing the lifting of the right adjoint, such that we generalise the two-stage construction above.

We show that, in this setting of an invertible step, the second stage of the two-stage construction preserves and reflects bisimilarity, under mild conditions. As a consequence, we recover the above-mentioned results on preservation and reflection of behavioural equivalence for probabilistic automata [12] for free from the abstract theory.[3] Another motivating example is that of coalgebras for the Vietoris functor on the category of Stone spaces: we obtain that bisimilarity is preserved and reflected by the forgetful functor, recovering the main result in [5].

In fact, the latter example is related to a coalgebraic presentation [36] of ultrafilter extensions, a standard construction in modal logic [6]. It fits within the general setting of invertible steps, but not directly in weak liftings, as it involves the category of Stone spaces (for the duality with Boolean algebras). However, if we move from Stone spaces to compact Hausdorff spaces, then the relevant weak lifting (or invertible step) arises precisely from the weak distributive law

[3] We focus on bisimilarity, but our setting allows for an easy argument that this coincides with behavioural equivalence in this and many related examples.

constructed by Garner [19]. The weak distributive law in *loc. cit.* thus gives rise to ultrafilter extensions in modal logic.

Finally, we include an example of an invertible step involving $\mathsf{Set}^{\mathsf{op}}$ instead of an Eilenberg-Moore category. Steps for adjunctions with opposite categories are a standard way of presenting the semantics of coalgebraic modal logic [40,32]. The included example shows the generality of the approach.

Outline. Section 2 presents (invertible) steps, the relation to weak liftings and distributive laws, and a range of examples. In Section 3 we recall the standard notion of coalgebraic bisimilarity, defined via relation lifting. Section 4 contains the main results on preservation and reflection of bisimilarity. In Section 5 we discuss applications and instances of these results. We discuss other notions of bisimulation, and future work, in Section 6.

2 Forward and Backward Steps

We briefly present the required theory of steps, first termed as such in [41]. This structure occurs already in work on coalgebraic modal logic [35,14,40,32,17,38] where a step gives the one-step semantics of a logic. In existing work, only what we call a *forward* step is considered. Here, we also speak of *backward* steps, being arrows in the opposite direction. In the sequel, such forward and backward steps will usually be each other's (one-sided) inverses, referred to as *invertible steps.*

Next, we recall how such steps give rise to liftings of functors between categories of coalgebras and further, when the adjunction underlying the steps can also be lifted to coalgebras [27]. Finally, we present examples of invertible steps from the literature, which we return to in later sections.

For a functor $B\colon \mathcal{C} \to \mathcal{C}$, a *coalgebra* is a pair (X, f) consisting of an object X and an arrow $f\colon X \to BX$. A homomorphism from (X, f) to (Y, g) is an arrow $h\colon X \to Y$ such that $g \circ h = Bh \circ f$. Coalgebras and homomorphisms between them form a category, denoted by $\mathsf{Coalg}(B)$, or $\mathsf{Coalg}_{\mathcal{C}}(B)$ if we wish to make the underlying category explicit.

The category of sets and functions is denoted by Set. For a monad T, we write $\mathcal{EM}(T)$ for the category of Eilenberg-Moore algebras. The powerset monad is denoted by $\mathcal{P}\colon \mathsf{Set} \to \mathsf{Set}$, given on objects by $\mathcal{P}(X) = \{S \mid S \subseteq X\}$, and the finitely-supported distribution monad by $\mathcal{D}\colon \mathsf{Set} \to \mathsf{Set}$, given by $\mathcal{D}(X) = \{\varphi\colon X \to [0,1] \mid \sum_{x\in X} \varphi(x) = 1,\ \mathrm{supp}(\varphi) \text{ finite}\}$ (see also [12]).

2.1 Invertible Steps

The basic setting of interest in this work consists of the following:

Definition 2.1. *Given an adjunction $P \dashv Q\colon \mathcal{D} \to \mathcal{C}$ and endofunctors $B\colon \mathcal{C} \to \mathcal{C}$ and $L\colon \mathcal{D} \to \mathcal{D}$ as in the diagram*

$$B \circlearrowright \mathcal{C} \underset{Q}{\overset{P}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathcal{D} \circlearrowleft L\,, \tag{1}$$

a (forward) step *is a natural transformation* $\delta\colon BQ \to QL$. *A* backward step *is simply a natural transformation* $\iota\colon QL \to BQ$ *going the other way. If, moreover,* $\delta \circ \iota = \mathsf{id}$ *then we call* δ *an* invertible step *(with right inverse* ι*). Finally, if* δ *witnesses an isomorphism then we call it an* isomorphic step.

Notice the asymmetry in the definition of invertible step: ι is always assumed to be a *right* inverse of δ. These invertible steps are the main focus of this paper. Examples are given below in Section 2.2.

Step-induced liftings There is a bijective correspondence between a step and its *mate* $\hat{\delta}\colon PB \to LP$ given by $PB \xrightarrow{PB\eta} PBQP \xrightarrow{P\delta P} PQLP \xrightarrow{\varepsilon LP} LP$ (see [37,31]). This mate and the backward step allow us to define liftings of P and Q to the categories of coalgebras for B and L.

Definition 2.2. *Given steps* $\delta\colon BQ \to QL$ *and* $\iota\colon QL \to BQ$*, the* step-induced coalgebra liftings $\overline{P}\colon \mathsf{Coalg}(B) \to \mathsf{Coalg}(L)$ *and* $\overline{Q}\colon \mathsf{Coalg}(L) \to \mathsf{Coalg}(B)$ *of* P *and* Q *are defined by*

$$f\colon X \to BX \quad \mapsto \quad \hat{\delta}_X \circ Pf\colon PX \to LPX \tag{2}$$

$$g\colon Y \to LY \quad \mapsto \quad \iota_Y \circ Qg\colon QY \to BQY \tag{3}$$

on objects and act as P *and* Q *on arrows. This is well-defined due to functoriality of* P *and* Q *and naturality of* $\hat{\delta}$ *and* ι.

It is shown in [27, Theorem 2.14] that, when δ and ι form an isomorphism, the adjunction $P \dashv Q$ lifts to an adjunction $\overline{P} \dashv \overline{Q}$ between the step-induced liftings. For our purposes it will be useful to split the isomorphism condition into the cases where ι is the left or right inverse of δ.

Lemma 2.3. *If* $\delta \circ \iota = \mathsf{id}$*, then the counit* $\varepsilon\colon PQ \to \mathsf{Id}$ *of the adjunction* $P \dashv Q$ *lifts to a natural transformation* $\overline{\varepsilon}\colon \overline{P}\,\overline{Q} \to \mathsf{Id}$*. If* $\iota \circ \delta = \mathsf{id}$*, then the unit* $\eta\colon \mathsf{Id} \to QP$ *of the adjunction lifts to a natural transformation* $\overline{\eta}\colon \mathsf{Id} \to \overline{Q}\,\overline{P}$.

The combination of these two liftings gives us the lifting of the adjunction.

Corollary 2.4. *If* δ *and* ι *form an isomorphism, then* $\overline{P} \dashv \overline{Q}$.

In such a situation, $\overline{Q}$ (being a right adjoint) preserves the final coalgebra for L (the limit of the empty diagram) when this exists. However, there are a number of known examples where the step is not an isomorphism; instead we only have a one-sided inverse. We consider, in particular, these invertible steps, and in the next subsection give a number of examples of this setting.

2.2 Steps from weak liftings, and other examples

Example 2.5. Our first example arises from the work of Garner, who shows that the Vietoris monad $\mathcal{V}$ on the category CHaus of compact Hausdorff spaces arises

as a so-called *weak lifting* of the powerset monad [19] (we discuss weak liftings in general after this example). For the definition of the Vietoris monad the reader is referred to [19, Sec. 2.3]. The category CHaus is equivalent to the Eilenberg-Moore category $\mathcal{EM}(\beta)$ of the ultrafilter monad β [39]. The weak lifting provided by Garner consists of natural transformations ι, δ, satisfying $\delta \circ \iota = \mathsf{id}$:

$$\mathcal{P} \circlearrowright \mathsf{Set} \underset{U}{\overset{\mathcal{F}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathcal{EM}(\beta) \circlearrowright \mathcal{V} \qquad\qquad U\mathcal{V} \xrightarrow{\iota} \mathcal{P}U \xrightarrow{\delta} U\mathcal{V} \tag{4}$$

where $\mathcal{F} \dashv U$ is the Eilenberg-Moore adjunction of β. Notice that δ is an invertible step, with right inverse ι. As shown by Garner, a component $\delta_X \colon \mathcal{P}UX \to U\mathcal{V}X$, sends each subset $S \in \mathcal{P}UX$ to its topological closure. The components of ι simply include the closed subsets into the powerset.

It turns out that this invertible step gives rise to ultrafilter extensions of Kripke frames. In modal logic, ultrafilter extensions [6,20,4] are a construction taking a Kripke frame (which we can see as a coalgebra for the powerset functor $\mathcal{P}$) with state space W and forming a new Kripke frame with states being ultrafilters over W. The central motivation for this is in "bisimilarity-somewhere-else" results: two states are modally equivalent iff they are bisimilar in the ultrafilter extension.

Now, the composition of the step-induced coalgebra liftings $\overline{\mathcal{F}} \colon \mathsf{Coalg}(\mathcal{P}) \to \mathsf{Coalg}(\mathcal{V})$ and $\overline{U} \colon \mathsf{Coalg}(\mathcal{V}) \to \mathsf{Coalg}(\mathcal{P})$, precisely yields the ultrafilter extension of a Kripke frame. The first stage $\overline{\beta}$ is the actual extension, which turns the Kripke frame into a $\mathcal{V}$-coalgebra. The second stage $\overline{U}$ turns this back into a Kripke frame, i.e., a powerset coalgebra in Set.

In [36], ultrafilter extensions are developed more generally for coalgebras for a functor $B \colon \mathsf{Set} \to \mathsf{Set}$, via the duality between Boolean algebras and Stone spaces. In fact, since both $\mathcal{V}$ and the left adjoint $\mathcal{F}$ restrict to the category Stone of Stone spaces, the invertible step δ, ι restricts to an invertible step in the restriction of the above adjunction to Stone.

In general, for monads S, T on a category $\mathcal{C}$, Garner [19] defines $\tilde{S} \colon \mathcal{EM}(T) \to \mathcal{EM}(T)$ to be a weak lifting of S if there are natural transformations

$$U\tilde{S} \xrightarrow{\iota} SU \xrightarrow{\delta} U\tilde{S} \tag{5}$$

with $\delta \circ \iota = \mathsf{id}$ and satisfying further axioms, where U denotes the forgetful functor from $\mathcal{EM}(T)$ to $\mathcal{C}$. They show that there is a bijective correspondence between weak distributive laws of T over S, and weak liftings of S to $\mathcal{EM}(T)$, in case idempotents in $\mathcal{C}$ split (which holds for Set). Here, we do not assume a monad structure on S (which is why the additional axioms are not relevant). In this case, a weak lifting is precisely an invertible step, where the underlying adjunction is an Eilenberg-Moore adjunction.

Example 2.6. In [11,12], a procedure is given for "determinising" probabilistic automata (PAs), which model systems with both non-determinism and probabilities, into belief state transformers. It was shown in [22] that this is an instance

of a more general determinisation procedure induced by a weak lifting, which in turn corresponds to a canonical weak distributive law.

Stated for a general monad T with the usual Eilenberg-Moore adjunction $\mathcal{F} \dashv U\colon \mathcal{EM}(T) \to \mathcal{C}$, this general determinisation procedure thus starts from an invertible step (weak lifting) $\delta\colon BU \to U\overline{B}$. This gives rise to a two-step process:

$$\mathsf{Coalg}_{\mathcal{C}}(BT) \xrightarrow{\overline{\mathcal{F}}} \mathsf{Coalg}_{\mathcal{EM}(T)}(\overline{B}) \xrightarrow{\overline{U}} \mathsf{Coalg}_{\mathcal{C}}(B) \tag{6}$$

where the second functor $\overline{U}$ is simply the step-induced lifting of U. The first is a variation of a step-induced lifting (notice that it takes BT-coalgebras rather than B-coalgebras as input), mapping a coalgebra $f\colon X \to BTX$ to $\mathcal{F}X \xrightarrow{\mathcal{F}f} \mathcal{F}B\mathcal{U}\mathcal{F}X \xrightarrow{\hat{\delta}_{\mathcal{U}\mathcal{F}X}} \overline{B}\mathcal{F}\mathcal{U}\mathcal{F}X \xrightarrow{\overline{B}\varepsilon_{\mathcal{F}X}} \overline{B}\mathcal{F}X$, where ε is the counit of the Eilenberg-Moore adjunction. In fact, this can be viewed as a step-induced lifting for BT which arises by composing δ and the counit, see [41].

We instantiate this to the Eilenberg-Moore adjunction of the distribution monad $\mathcal{D}$, where $\mathcal{P}_c$ is the convex powerset monad:

$$\mathcal{P} \circlearrowright \mathsf{Set} \underset{U}{\overset{\mathcal{F}}{\underset{\curvearrowright}{\overset{\curvearrowleft}{\bot}}}} \mathcal{EM}(\mathcal{D}) \circlearrowleft \mathcal{P}_c \tag{7}$$

We take $\mathcal{P}_c(X)$ to have as underlying set $\{S \subseteq X \mid S \text{ convex}\}$ following [22]. This matches the usage of $\mathcal{P}_{ne} + 1$ and $\mathcal{P}_c + 1$ in [12], where $\mathcal{P}_{ne}$ and $\mathcal{P}_c$ are defined to exclude the empty set. A subset is convex if it is closed under convex combinations (see [12] for details). Further, the category $\mathcal{EM}(\mathcal{D})$ is equivalent to the category of convex algebras and convex maps.

It is explained in [22, Sec. 5] that we have an invertible step in the setting of Eq. (7), which sends a subset X to its convex hull (the smallest convex set containing X) and that the lifting $\overline{\mathcal{F}}$ of (6) then gives the transformation of a probabilistic automaton into a belief state transformer in the category $\mathcal{EM}(\mathcal{D})$. The second step is then to transfer the obtained belief state transformer back to Set with the step-induced lifting of U. As shown in [12] and later recovered from our abstract theory (Section 5), this yields a system with the same behaviour. In fact, this is done for automata with labels, i.e., for the functors $\mathcal{P}^L$ and $\mathcal{P}_c^L$ with L a set of labels. The weak lifting we will require in this context is given in [21].

Example 2.7. The following example from automata and languages considers a dual adjunction $P \dashv Q\colon \mathcal{D}^{\mathsf{op}} \to \mathcal{C}$. One motivation to discuss this kind of example stems from coalgebraic modal logic where $\mathcal{C}$ commonly is some category of 'spaces' and $\mathcal{D}$ commonly is a category of 'algebras' [32]. The setup is as follows:

$$B \circlearrowright \mathsf{Set} \underset{2^-}{\overset{2^-}{\underset{\curvearrowright}{\overset{\curvearrowleft}{\bot}}}} \mathsf{Set}^{\mathsf{op}} \circlearrowleft L \qquad\qquad 2^L \xrightarrow{\iota} B(2^-) \xrightarrow{\delta} 2^L \tag{8}$$

Here, we have $BX = 2 \times (\mathcal{P}X)^{\Sigma}$ and $LX = 1 + \Sigma \times X$ for a fixed alphabet Σ. The step δ is given by

$$\delta(i, \xi) = \{\mathsf{inl}(*) \mid i = 1\} \cup \{\mathsf{inr}(a, x) \mid a \in \Sigma, x \in \bigcup \xi(a)\} \tag{9}$$

This step δ is invertible, e.g., by ι as in Eq. (10).

$$\iota(u) = (1 \text{ iff } \mathsf{inl}(*) \in u, a \mapsto \{v \mid \{(a,x) \mid x \in v\} \subseteq u\}) \tag{10}$$

A B-coalgebra is a non-deterministic automaton. An L-coalgebra in $\mathsf{Set}^{\mathsf{op}}$ is an algebra $X \leftarrow 1 + \Sigma \times X$ in Set, which can be seen as specifying the initial state and transition structure of a deterministic automaton. From this point of view, the coalgebra lifting $\overline{Q}\colon \mathsf{Coalg}(L) \to \mathsf{Coalg}(B)$ can be seen as first reversing, and then performing a powerset construction. The specific powerset construction might depend on the chosen right inverse ι, as it is not unique. For ι as in (10), for example, $u \xrightarrow{a} v$ in $\overline{Q}(\mathcal{A})$ if and only if each state in v is reachable from a state in u via an a-transition in the reverse of $\mathcal{A}$.

In Section 5 we return to these examples and show how we can apply the techniques from Section 4 to obtain preservation and reflection of bisimilarity.

3 Relations, Liftings and Coalgebraic Bisimulations

We recall the standard notion of coalgebraic bisimulation defined via relation lifting, broadly following [30,28]. Note, we will use some terminology from the theory of fibrations to allow us to be more concise and many of the coming results can be generalised to a larger class of fibrations, but knowledge of fibrations is not required as we give a self-contained presentation of the fibration of relations.

We make the following assumptions for the remainder of the paper:

Assumption 3.1. *We assume categories $\mathcal{C}, \mathcal{D}$ with all finite limits, and factorisation systems $(\mathcal{E}_1, \mathcal{M}_1)$, $(\mathcal{E}_2, \mathcal{M}_2)$ respectively for which $\mathcal{M}_1 = \mathsf{Mono}_{\mathcal{C}}, \mathcal{M}_2 = \mathsf{Mono}_{\mathcal{D}}$ and for any left adjoint functor $P\colon \mathcal{C} \to \mathcal{D}$ we have $P(\mathcal{E}_1) \subseteq \mathcal{E}_2$.*

We assume finite limits mainly for binary products and pullbacks to allow the definitions of relations and inverse images. The assumptions that maps in $\mathcal{M}$ are mono means that pullbacks of abstract monos and factorisation both yield monos, which represent subobjects. The final condition specifies that left adjoints preserve abstract epis. This is required in Section 4.2 and holds, e.g., when the involved categories possess a $(\mathsf{RegEpi}, \mathsf{Mono})$-factorisation system [16,2], as in all our examples from Sections 2.2 and 5.

For a category $\mathcal{C}$ satisfying the above, the category $\mathsf{Rel}(\mathcal{C})$ consists of:

- Objects of $\mathsf{Rel}(\mathcal{C})$ are subobjects $R \rightarrowtail X \times X$ of the binary product of the object X with itself;
- A map $R \rightarrowtail X \times X \to S \rightarrowtail Y \times Y$ in $\mathsf{Rel}(\mathcal{C})$ consists of a map $u\colon X \to Y$ in $\mathcal{C}$ such that there is the following commutative diagram

$$\begin{array}{ccc} R & \dashrightarrow & S \\ \downarrow & & \downarrow \\ X \times X & \xrightarrow{u \times u} & Y \times Y \end{array} \tag{11}$$

In Set, these are subsets of the binary product of underlying sets as usual, and maps between relations constitute maps between the products sending R to S, i.e., $x\,R\,y$ implies $u(x)\,S\,u(y)$. Objects of $\mathsf{Rel}(\mathsf{Stone})$ are closed relations, as the image of a mono representing a subobject is homeomorphic to its domain, and images of continuous functions are compact and thus closed. In the case of an Eilenberg-Moore category for a monad T, objects of $\mathsf{Rel}(\mathcal{EM}(T))$ are congruences, as the map into the product is an algebra morphism.

Remark 3.2. A note on notation: we use $\twoheadrightarrow$ for epis and $\rightarrowtail$ for monos and the subobjects they represent. We use $\longrightarrow$ for abstract epis and $\triangleright\!\!\rightarrow$ for abstract monos, i.e., maps in $\mathcal{E}$ and $\mathcal{M}$ respectively.

Using the factorisation system on $\mathcal{D}$, we lift a functor $F\colon \mathcal{C} \to \mathcal{D}$ to a functor $\mathsf{Rel}(F)\colon \mathsf{Rel}(\mathcal{C}) \to \mathsf{Rel}(\mathcal{D})$. The action on objects is given by the factorisation

$$\begin{array}{ccccc} FR & \xrightarrow{Fr} & F(X \times X) & \xrightarrow{\langle F\pi_1, F\pi_2\rangle} & FX \times FX \\ & \searrow^{e} & \mathsf{Rel}(F)(R) & \triangleright\!\!\rightarrow_{m} & \nearrow \end{array} \tag{12}$$

The action on arrows is defined by orthogonality. The resulting functor $\mathsf{Rel}(F)$ is a lifting in the sense that the following diagram commutes

$$\begin{array}{ccc} \mathsf{Rel}(\mathcal{C}) & \xrightarrow{\mathsf{Rel}(F)} & \mathsf{Rel}(\mathcal{D}) \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle q} \\ \mathcal{C} & \xrightarrow{F} & \mathcal{D} \end{array} \tag{13}$$

where $p\colon \mathsf{Rel}(\mathcal{C}) \to \mathcal{C}$ sends a relation $R \rightarrowtail X \times X$ to the object X, and similarly for q. We say (following the terminology of fibrations) that the relation R is above the object X and a map between relations is above the map u from Eq. (11). Note that commutativity of diagram (13) expresses that $\mathsf{Rel}(F)$, applied to a relation $R \rightarrowtail X \times X$ on X, yields a relation on FX.

Given a category of relations $\mathsf{Rel}(\mathcal{C})$, called the total category, the subcategory (also called a fibre) Rel_X consists of objects $R \rightarrowtail X \times X$ and maps above the identity on X. For relations in Set, such maps are inclusions of relations. In general, these maps are unique, and writing $R \leq S$ iff there is an arrow from R to S turns the fibre into a poset. A relation lifting $\mathsf{Rel}(F)$ can be restricted to the fibres to give a functor $\mathsf{Rel}(F)_X\colon \mathsf{Rel}_X \to \mathsf{Rel}_{FX}$. Since Rel_X and Rel_{FX} are posetal categories, $\mathsf{Rel}(F)_X$ can be viewed as a monotone map.

For a map $f\colon X \to Y$ in $\mathcal{C}$, we have the direct image and inverse image functors $\coprod_f\colon \mathsf{Rel}_X \to \mathsf{Rel}_Y$ and $f^*\colon \mathsf{Rel}_Y \to \mathsf{Rel}_X$. For relations on sets, we have $\coprod_f(R \subseteq X \times X) = \{(f(x), f(y)) \mid (x, y) \in R\}$ and $f^*(S \subseteq Y \times Y) = \{(x, y) \in X \times X \mid (f(x), f(y)) \in S\}$. More generally, they are obtained as the factorisation and pullback in the left and right diagram below respectively

$$\begin{array}{ccc} R & \longrightarrow & \coprod_f(R) \\ {\scriptstyle r}\downarrow & & \downarrow{\scriptstyle \coprod_f(r)} \\ X \times X & \xrightarrow{f\times f} & Y \times Y \end{array} \qquad\qquad \begin{array}{ccc} f^*(S) & \longrightarrow & S \\ {\scriptstyle f^*(s)}\downarrow & \lrcorner & \downarrow{\scriptstyle s} \\ X \times X & \xrightarrow{f\times f} & Y \times Y \end{array} \tag{14}$$

It can further be shown that $\coprod_f \dashv f^*$. We say that $\mathsf{Rel}(F)\colon \mathsf{Rel}(\mathcal{C}) \to \mathsf{Rel}(\mathcal{D})$ preserves inverse images if $\mathsf{Rel}(F)_X \circ f^* = (Ff)^* \circ \mathsf{Rel}(F)_Y$.

In this context, *a bisimulation for a B-coalgebra $f\colon X \to BX$* is a post-fixed point of the endofunctor $f^* \circ \mathsf{Rel}(B)_X\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$, i.e., a relation $R \rightarrowtail X \times X$ such that $R \leq f^* \circ \mathsf{Rel}(B)_X(R)$. Bisimilarity is then obtained as the greatest fixed point $\nu(f^* \circ \mathsf{Rel}(B)_X)$, if it exists. In Set a bisimulation is a relation R such that $R \subseteq (f \times f)^{-1}(\mathsf{Rel}(B)(R))$, i.e., if $x\,R\,y$ then $f(x)\ \mathsf{Rel}(B)(R)\ f(y)$.

4 Preserving and Reflecting Bisimilarity

In this section we show that, in the presence of an invertible step, bisimilarity is preserved and reflected by the step-induced lifting of the right adjoint, given some further mild conditions. This allows us to recover a number of existing results for concrete instances (Section 5).

Our approach is as follows:

- In Section 4.1, we make precise what it means for a monotone map to preserve and reflect bisimulations;
- In Section 4.2, we obtain conditions which ensure that the step-induced lifting of the right adjoint to bisimulations preserves and reflects bisimulations/bisimilarity.

Throughout this section we assume categories $\mathcal{C}$ and $\mathcal{D}$ as in Assumption 3.1, and an invertible step $\delta\colon BQ \to QL$ with right inverse $\iota\colon QL \to BQ$ (and P, Q, B, L as in Definition 2.1).

4.1 Preservation and reflection

We now make precise what it means for a monotone map h to preserve and reflect bisimulations. This will be instantiated to bisimulations, captured abstractly as post-fixed points of a monotone map $f\colon \Gamma \to \Gamma$ on a poset Γ, which typically consists of relations (Section 3). These are compared against a second type of bisimulations, modelled as post-fixed points of another monotone map $g\colon \Delta \to \Delta$. This motivates the following definition.

Definition 4.1. *Let Γ and Δ be posets, and $f\colon \Gamma \to \Gamma$, $g\colon \Delta \to \Delta$ monotone maps. A monotone map $h\colon \Gamma \to \Delta$* preserves post-fixed points *if $x \leq f(x)$ implies $h(x) \leq g(h(x))$. It* reflects post-fixed points *if the converse implication holds.*

In the step setting of Eq. (1), bisimulations for B- and L-coalgebras can be represented as post-fixed points of monotone maps on posets of relations as in Section 3. More concretely:

- Bisimulations for an L-coalgebra $f\colon X \to LX$ are post-fixed points of the monotone map $f^* \circ \mathsf{Rel}(L)_X\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$;
- Bisimulations for the B-coalgebra $\iota_X \circ Qf\colon QX \to BQX$ resulting from the application of the step-induced lifting of Q are post-fixed points of $(\iota_X \circ Qf)^* \circ \mathsf{Rel}(B)_{QX}\colon \mathsf{Rel}_{QX} \to \mathsf{Rel}_{QX}$.

The two can be compared via the restriction $\mathsf{Rel}(Q)_X \colon \mathsf{Rel}_X \to \mathsf{Rel}_{QX}$ of the functor $\mathsf{Rel}(Q)$. Indeed, our main objective is to show that in the presence of an invertible step, $\mathsf{Rel}(Q)_X$ preserves and reflects post-fixed points representing bisimulations, and that it maps the greatest fixed point in Rel_X (bisimilarity on f) to the greatest fixed point in Rel_{QX} (bisimilarity on $\iota_X \circ Qf$). In this context we speak about *preservation and reflection of bisimulations/bisimilarity.*

4.2 Proof of preservation and reflection

We are now ready to prove preservation and reflection of bisimilarity, in the sense described in the previous subsection. First, the following basic lemma provides a method of showing preservation and reflection of post-fixed points, which will be useful for our purposes.

Lemma 4.2. *Let Γ and Δ be posets, and $f \colon \Gamma \to \Gamma$, $g \colon \Delta \to \Delta$ and $h \colon \Gamma \to \Delta$ monotone maps. Suppose that h has a left (lower) adjoint $k \colon \Delta \to \Gamma$, and the equality $gh = hf$ holds. Then h maps the greatest fixed point of f to the greatest fixed point of g, when these exist; h preserves post-fixed points; and if h is order-reflecting, then h reflects post-fixed points.*

Categorically speaking, the equality $gh = hf$ is an isomorphic step. Instantiated to our setting of interest, Lemma 4.2 gives us a method for proving preservation and reflection of bisimilarity: it suffices to show each of the following.

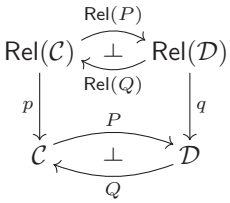
1. A left adjoint for $\mathsf{Rel}(Q)_X$ (Lemma 4.7).
2. The equality $(\iota_X \circ Qf)^* \circ \mathsf{Rel}(B)_{QX} \circ \mathsf{Rel}(Q)_X = \mathsf{Rel}(Q)_X \circ f^* \circ \mathsf{Rel}(L)_X$ (Theorem 4.9).
3. Order-reflection of $\mathsf{Rel}(Q)_X$ (assumption; discussed at the end of this section).

To obtain the required adjunction between the fibres Rel_X and Rel_{QX}, we first establish the adjunction $\mathsf{Rel}(P) \dashv \mathsf{Rel}(Q)$ between the total relation categories. Given Theorem 3.1, we can lift the unit and counit of the adjunction $P \dashv Q$, using the transformations constructed in the following lemma.

Lemma 4.3. *Let $F \colon \mathcal{C} \to \mathcal{D}$ and $G \colon \mathcal{D} \to \mathcal{E}$ be functors, with $\mathsf{Rel}(F) \colon \mathsf{Rel}(\mathcal{C}) \to \mathsf{Rel}(\mathcal{D})$ and $\mathsf{Rel}(G) \colon \mathsf{Rel}(\mathcal{D}) \to \mathsf{Rel}(\mathcal{E})$ the corresponding relation liftings. Then we have a natural transformation $\mathsf{Rel}(GF) \to \mathsf{Rel}(G)\,\mathsf{Rel}(F)$. Further, if G preserves abstract epis, then there is also a natural transformation $\mathsf{Rel}(G)\,\mathsf{Rel}(F) \to \mathsf{Rel}(GF)$. Also, the constructed transformations are above the identity.*

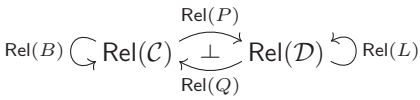
We note that the first part is in [28, Exercise 4.4.6] and the result is proved for Set endofunctors in [9, Lemma 14.1]. This allows the lifting of the adjunction, which we note may also be obtainable from results on fibred adjunctions in [30,26], but a direct proof is quite straightforward; the main idea is to use Lemma 4.3 together with preservation of abstract epis by P.

Lemma 4.4. *The adjunction $P \dashv Q \colon \mathcal{D} \to \mathcal{C}$ lifts to relations, i.e., the following diagram is commutative, and the unit and counit of the upper adjunction are above the unit and counit of $P \dashv Q$.*



(15)

The relation lifting defined in Section 3 allows us to define endofunctors $\mathsf{Rel}(B)$, $\mathsf{Rel}(L)$ in the context of the above adjunction:



(16)

In this setting, we may try to lift the step δ or its converse ι to this adjunction. It turns out that δ always lifts. For ι, there is a sufficient condition which is independent of ι itself: that Q preserves abstract epis. In both cases, this result follows essentially from Lemma 4.3.

Proposition 4.5. *For a forward step δ and backward step ι, we have:*

1. *δ lifts to relations, i.e., there exists a natural transformation $\overline{\delta} \colon \mathsf{Rel}(B) \circ \mathsf{Rel}(Q) \to \mathsf{Rel}(Q) \circ \mathsf{Rel}(L)$ above δ.*
2. *If Q preserves abstract epis, then ι lifts to relations, i.e., there exists a natural transformation $\overline{\iota} \colon \mathsf{Rel}(Q) \circ \mathsf{Rel}(L) \to \mathsf{Rel}(B) \circ \mathsf{Rel}(Q)$ above ι.*

The condition that Q preserves abstract epis holds, e.g., in case it is the forgetful functor in an adjunction monadic over Set. This is because Eilenberg-Moore categories of monads on Set have $(\mathsf{RegEpi}, \mathsf{Mono})$-factorisation systems, and the forgetful functor sends regular epis to epis in Set as discussed in [13, Example 2.3]. It also holds in the Stone-Set case, as Stone is a reflective subcategory of CHaus (which is equivalent to the category of algebras for the ultrafilter monad).

The lifted steps $\overline{\delta}$ and $\overline{\iota}$ give step-induced liftings of $\mathsf{Rel}(P)$ and $\mathsf{Rel}(Q)$ between $\mathsf{Coalg}(\mathsf{Rel}(B))$ and $\mathsf{Coalg}(\mathsf{Rel}(L))$. Since bisimulations can be equivalently presented as coalgebras for $\mathsf{Rel}(B)$ and $\mathsf{Rel}(L)$, these liftings can be used to capture preservation of bisimulations. But it is less obvious what reflection means in this context and how to prove it. For reflection of bisimulations by $\mathsf{Rel}(Q)$, we turn our attention to the fibres, as described in the beginning of this section.

As a consequence of Proposition 4.5 and of $\delta \circ \iota = \mathsf{id}$, we obtain the following result, which will later be used in the construction of a step on the fibres.

Lemma 4.6. *Let δ be an invertible step with right inverse ι, and suppose Q preserves abstract epis. Then $\mathsf{Rel}(Q)_{LX} \circ \mathsf{Rel}(L)_X = \iota_X^* \circ \mathsf{Rel}(B)_{QX} \circ \mathsf{Rel}(Q)_X$.*

Adjoining the fibres Next, we construct an adjunction between the fibres Rel_X and Rel_{QX}. The usual restriction $\mathsf{Rel}(Q)_X$ of $\mathsf{Rel}(Q)$ to the fibre Rel_X will be the right adjoint, similarly to the adjunction obtained earlier. To map back into the fibre Rel_X, we post-compose $\mathsf{Rel}(P)_{QX}$ with $\coprod_\varepsilon$, the direct image functor obtained from the counit of the adjunction $\mathsf{Rel}(P) \dashv \mathsf{Rel}(Q)$. We note the similarity with results on fibred adjunctions in [30], where only adjunctions over a single base category are considered.

Lemma 4.7. *We have an adjunction* $\coprod_\varepsilon \circ \mathsf{Rel}(P)_{QX} \dashv \mathsf{Rel}(Q)_X \colon \mathsf{Rel}_X \to \mathsf{Rel}_{QX}$.

The above lemma fulfils the first proof obligation stated in the beginning of Section 4.2. It now remains to show the second proof obligation, i.e., that we have an isomorphic step in the following setting:

$$(\iota_X \circ Qf)^* \circ \mathsf{Rel}(B)_{QX} \circlearrowright \mathsf{Rel}_{QX} \underset{\mathsf{Rel}(Q)_X}{\overset{\coprod_\varepsilon \circ \mathsf{Rel}(P)_{QX}}{\rightleftarrows}} \mathsf{Rel}_X \circlearrowright f^* \circ \mathsf{Rel}(L)_X \tag{17}$$

To this end, we first show that $\mathsf{Rel}(Q)$ preserves inverse images, using the fact that we can obtain inverse images as pullbacks inside the category of relations. Since $\mathsf{Rel}(Q)$ is a right adjoint, it preserves these pullbacks.

Lemma 4.8. $\mathsf{Rel}(Q)$ *preserves inverse images.*

We are now ready to show the existence of the required isomorphic step.

Theorem 4.9. *If Q preserves abstract epis, then for any L-coalgebra (X, f):*

$$(\iota_X \circ Qf)^* \circ \mathsf{Rel}(B)_{QX} \circ \mathsf{Rel}(Q)_X = \mathsf{Rel}(Q)_X \circ f^* \circ \mathsf{Rel}(L)_X \tag{18}$$

Proof. We have

$$\begin{aligned}
(\iota_X \circ Qf)^* \circ \mathsf{Rel}(B)_{QX} \circ \mathsf{Rel}(Q)_X &= (Qf)^* \circ \iota_X^* \circ \mathsf{Rel}(B)_{QX} \circ \mathsf{Rel}(Q)_X && (19)\\
&= (Qf)^* \circ \mathsf{Rel}(Q)_{LX} \circ \mathsf{Rel}(L)_X && (20)\\
&= \mathsf{Rel}(Q)_X \circ f^* \circ \mathsf{Rel}(L)_X && (21)
\end{aligned}$$

where Eq. (19) is an application of a basic fact on inverse images (technically, that the poset fibration of relations is split), Eq. (20) holds by Lemma 4.6, and Eq. (21) holds by Lemma 4.8. □

We now reach our main result on preservation and reflection of bisimulations and bisimilarity by $\mathsf{Rel}(Q)_X$.

Theorem 4.10. *Let (X, f) be an L-coalgebra. Suppose that Q preserves abstract epis. Then $\mathsf{Rel}(Q)_X$ maps bisimilarity on (X, f) (when it exists) to bisimilarity on $\overline{Q}(X, f)$. Further, $\mathsf{Rel}(Q)_X$ preserves bisimulations and, if it is order-reflecting, also reflects bisimulations.*

Proof. We have seen in Lemma 4.7, that $\mathsf{Rel}(Q)_X$ has a left adjoint, and in Theorem 4.9, that in this setting we have an isomorphic step. The result now follows from Lemma 4.2. □

While this result is formulated in terms of $\mathsf{Rel}(Q)_X$, we will also speak of simply $\overline{Q}$ preserving and reflecting both bisimulations and bisimilarity.

As a special case of Theorem 4.10, we recover (a version of) the following existing result found in [42,3,11,12].

Lemma 4.11. *Assume functors $B, L\colon \mathcal{C} \to \mathcal{C}$, and a natural transformation $\iota\colon L \to B$. Then the functor $\overline{\mathsf{Id}}\colon \mathsf{Coalg}(L) \to \mathsf{Coalg}(B)$ defined by $(X, f) \mapsto (X, \sigma_X \circ f)$ on objects and identity on morphisms, preserves bisimulations. If additionally ι has a left inverse, $\overline{\mathsf{Id}}$ reflects bisimulations.*

We briefly turn to the condition of order-reflectingness. As we are often interested in cases where the right adjoint is a forgetful functor in the context of an Eilenberg-Moore adjunction, it is useful to state the following.

Lemma 4.12. *For a monad T with forgetful functor $U\colon \mathcal{EM}(\mathcal{T}) \to \mathcal{C}$, the (restricted) lifting $\mathsf{Rel}(U)_X$ is an order-reflecting map.*

If $\mathcal{C} = \mathsf{Set}$ in the above lemma, then $\mathsf{Rel}(U)_X$ is just the inclusion of the poset of congruences Rel_X on an algebra X into the poset of all relations on its carrier.

In that case, we can also use the above to show preservation and reflection of *behavioural equivalence*. Two states of a coalgebra (in Set) are behaviourally equivalent if they can be identified by some coalgebra homomorphism. This can be captured more abstractly using kernel bisimulations (see, e.g., [44]). Since U is assumed to be a forgetful functor to Set, we simply define preservation and reflection of behavioural equivalence by $\overline{U}$ to mean that for any two states x, y of an L-coalgebra (X, f), x and y are behaviourally equivalent (for (X, f)) if and only if they are behaviourally equivalent for $\overline{U}(X, f)$.

It turns out that, in our setting, coincidence of bisimilarity and behavioural equivalence for L-coalgebras reduces to coincidence for B-coalgebras. This is stated in the following lemma; the essence is that $\overline{U}$ is easily shown to preserve behavioural equivalence.

Lemma 4.13. *For a monad T, consider the Eilenberg-Moore adjunction $\mathcal{F} \dashv U\colon \mathcal{EM}(T) \to \mathsf{Set}$ with functors $L\colon \mathcal{EM}(T) \to \mathcal{EM}(T)$ and $B\colon \mathsf{Set} \to \mathsf{Set}$, and an invertible step $\delta\colon BU \to UL$. Further suppose that $\overline{U}$ preserves and reflects bisimilarity, and that B preserves weak pullbacks. Then bisimilarity and behavioural equivalence for L-coalgebras coincide (and hence, $\overline{U}$ preserves and reflects behavioural equivalence).*

Remark 4.14. We conclude with a brief exploration of preservation and reflection by the restriction of the *left* adjoint $\mathsf{Rel}(P)_X$, in the setting of

$$f^* \circ \mathsf{Rel}(B)_X \circlearrowright \mathsf{Rel}_X \xrightarrow{\mathsf{Rel}(P)_X} \mathsf{Rel}_{PX} \circlearrowleft (\hat{\delta}_X \circ Pf)^* \circ \mathsf{Rel}(L)_{PX} \tag{22}$$

with $f\colon X \to BX$ a B-coalgebra in this case. Here, we can obtain a backward step $\mathsf{Rel}(P)_X \circ f^* \circ \mathsf{Rel}(B)_X \leq (\hat{\delta}_X \circ Pf)^* \circ \mathsf{Rel}(L)_{PX} \circ \mathsf{Rel}(P)_X$ which means that we can lift $\mathsf{Rel}(P)_X$ to bisimulations, so that these are preserved. However, we cannot obtain a forward step in this context, thus reflection will not hold. This is illustrated, e.g., by the example of ultrafilter extensions, where the ultrafilter monad β certainly does not reflect bisimulations: in general, in the ultrafilter extension more states will be bisimilar.

5 Applications

Now that we have obtained conditions for the preservation and reflection of bisimilarity, we return to the examples of Section 2.2. We will show how a number of existing non-trivial results can be recovered in a concise way. Further, the Set-Stone adjunction used in the first example is known to not be monadic, and so outside the scope of weak liftings, which indicates the generality of our results.

Ultrafilter Extensions and Vietoris bisimulations In Example 2.5, we have seen how the construction of ultrafilter extensions can be obtained from an invertible step, which arises from a weak lifting described by Garner. In the current treatment of reflection and preservation of bisimilarity, we focus on the restriction of this invertible step to the category Stone.

This brings us in line with [5], where a comparison is made between bisimilarity for the Vietoris functor $\mathcal{V}\colon \mathsf{Stone} \to \mathsf{Stone}$ and bisimilarity for the powerset functor $\mathcal{P}\colon \mathsf{Set} \to \mathsf{Set}$, called Vietoris-bisimilarity and Kripke-bisimilarity respectively in *op. cit.* More precisely, for a $\mathcal{V}$-coalgebra (X, f), Kripke bisimilarity is bisimilarity on $\overline{U}(X, f)$, where $\overline{U}$ is the step-induced lifting of the forgetful functor $U\colon \mathsf{Stone} \to \mathsf{Set}$. Vietoris bisimilarity is simply bisimilarity on the coalgebra (X, f) itself.

We consider the following results from [5]:

1. The relation liftings of $\mathcal{P}$ and $\mathcal{V}$ coincide for closed subsets [5, Prop 3.4]
2. Vietoris bisimulations are equivalently closed Kripke bisimulations [5, Thm 3.6]
3. The closure of a Kripke bisimulation is a Vietoris bisimulation [5, Thm 5.2]
4. Vietoris- and Kripke-bisimilarity are equivalent [5, Cor 3.10]

From the above discussion, we see that these results fit into the setting of Section 4, so that they can be recovered using our results on the preservation and reflection of bisimilarity as follows:

1. This follows from the equality of Lemma 4.6, as the action of ι^* is exactly the restriction to closed subsets. We can apply this lemma as U preserves abstract epis, due to the same argument as for adjunctions monadic over Set (see the discussion after Proposition 4.5), as Stone is also a regular category.
2. For this, we use preservation and reflection of bisimulations by the (restricted) relation lifting $\mathsf{Rel}(U)_X\colon \mathsf{Rel}_X \to \mathsf{Rel}_{UX}$ of the forgetful functor, which is simply the inclusion of the poset of closed relations on a Stone space X to that of all relations on the underlying set.

Indeed, preservation and reflection by $\mathsf{Rel}(U)_X$ follows from Theorem 4.10. We have seen that $U\colon \mathsf{Stone} \to \mathsf{Set}$ preserves abstract epis, so it only remains to check that $\mathsf{Rel}(U)_X$ is order-reflecting. This holds because Stone is a reflective (i.e. full) subcategory of CHaus, which is monadic over Set.

3. The left adjoint $\coprod_{\varepsilon} \circ \mathsf{Rel}(\beta)_{UX}$ of Lemma 4.7 gives the closure of a relation. Its lifting to bisimulations (cf. Remark 4.14) yields the desired result.
4. This holds since $\mathsf{Rel}(U)_X$ maps the greatest fixed point of $(\iota_X \circ Uf)^* \circ \mathsf{Rel}(\mathcal{P})_{UX}$ to that of $f^* \circ \mathsf{Rel}(\mathcal{V})_X$, i.e., it preserves and reflects bisimilarity.

PAs and Belief State Transformers As discussed in Example 2.6, we can determinise a PA to a coalgebra for the convex powerset functor $\mathcal{P}_c\colon \mathcal{EM}(\mathcal{D}) \to \mathcal{EM}(\mathcal{D})$ using a lifting of $\mathcal{F}\colon \mathsf{Set} \to \mathcal{EM}(\mathcal{D})$. The step-induced lifting of the corresponding forgetful functor $U\colon \mathcal{EM}(\mathcal{D}) \to \mathsf{Set}$ maps the $\mathcal{P}_c$-coalgebra back into Set, but we must take care that this does not change its behaviour. What we can do now, is show that bisimilarity is preserved and reflected.

Once we know this, we can apply Lemma 4.13 to show the coincidence of bisimilarity and behavioural equivalence in the case of the convex powerset functor on $\mathcal{EM}(\mathcal{D})$ and the powerset functor on Set as this preserves weak pullbacks. This coincidence is relevant for the generalisation of the corresponding results of [12] (restricted to the convex powerset functor), which are formulated in terms of behavioural equivalence. As mentioned in Example 2.6, the weak lifting we require to cover automata with labels can be found in [21]. Consider the following:

1. The lifting of the forgetful functor $U\colon \mathcal{EM}(\mathcal{D}) \to \mathsf{Set}$ preserves and reflects behavioural equivalence on $\mathcal{P}_c^L$-coalgebras [12, Proposition 6.6].
2. A relation R is a kernel bisimulation for a $\mathcal{P}_c^L$-coalgebra $(\mathbb{S}, c)$ in $\mathcal{EM}(\mathcal{D})$ iff it is a kernel bisimulation for $\overline{U}(\mathbb{S}, c)$ and also a congruence.

Again, we can apply the results of Section 4 to recover these results. In fact, in [12, Proposition 6.5], the second result is proved more generally, namely for settings where a so-called *lax lifting* exists rather than the weak lifting we require.

1. We have seen that U preserves abstract epis as the adjunction in question is monadic over Set. This allows us to apply Theorem 4.10 so that $\overline{U}$ indeed preserves and reflects bisimulations, and the relevant lifting preserves and reflects bisimilarity. From Lemma 4.13, it follows that $\overline{U}$ also preserves and reflects behavioural equivalence.
2. Assuming $(\mathbb{S}, c)$ is a $\mathcal{P}_c^L$-coalgebra, this follows from Lemma 4.13 together with the previous item, and the fact that bisimulations in Eilenberg-Moore categories are congruences.

Automata For a different instance, we revisit Example 2.7 and consider the basic adjunction $P \dashv Q\colon \mathcal{D}^{\mathsf{op}} \to \mathcal{C}$. As a general remark, we note that if $\mathcal{D}$ admits a factorization system $(\mathcal{E}, \mathcal{M})$ with $\mathcal{E}$ a class of epis, and $\mathcal{M}$ a class of monos, then $(\mathcal{M}, \mathcal{E})$ forms a factorization system for $\mathcal{D}^{\mathsf{op}}$, with $\mathcal{M}$ a class of epis *in* $\mathcal{D}^{\mathsf{op}}$, and $\mathcal{E}$ a class of monos *in* $\mathcal{D}^{\mathsf{op}}$. We can explicitly describe $\mathsf{Rel}(\mathcal{D}^{\mathsf{op}})$ as follows:

- Objects of $\mathsf{Rel}(\mathcal{D}^{\mathsf{op}})$ are quotients $X + X \twoheadrightarrow E$ of $X + X$;
- A map $X + X \twoheadrightarrow E \to Y + Y \twoheadrightarrow F$ consists of a map $u\colon Y \to X$ in $\mathcal{D}$ such that there is the following commutative diagram

$$\begin{array}{ccc} E & \dashleftarrow & F \\ \uparrow\!\!\uparrow & & \uparrow\!\!\uparrow \\ X + X & \xleftarrow[u+u]{} & Y + Y \end{array} \tag{23}$$

In the case $\mathcal{D} = \mathsf{Set}$, $\mathcal{E} = \mathsf{Epi}$ and $\mathcal{M} = \mathsf{Mono}$. Further, every epi $e\colon X + X \twoheadrightarrow E$ is isomorphic to an epi of the form $X + X \twoheadrightarrow (X + X)/{\sim}$ with $\sim$ an equivalence relation on $X + X$. This gives us an equivalent description of $\mathsf{Rel}(\mathsf{Set}^{\mathsf{op}})$:

- Objects of $\mathsf{Rel}(\mathsf{Set}^{\mathsf{op}})$ are equivalence relations $\sim$ on $X + X$ for a set X;
- A map $\sim \subseteq (X + X)^2 \to \approx \subseteq (Y + Y)^2$ consists of a map $u\colon Y \to X$ such that if $j(y) \approx j'(y')$, then $j(u(y)) \sim j'(u(y'))$, with j, j' arbitrary coproduct inclusions.

In particular, we see that the fibre over a set X consists of all equivalence relations on $X + X$, ordered by *reverse* inclusion. Reindexing along a map $u\colon X \leftarrow Y$ maps an equivalence relation $\approx$ on $Y + Y$ to the least equivalence relation $\sim$ on $X + X$, such that $j(u(y)) \sim j'(u(y'))$ for all $j(y) \approx j'(y')$.

Focusing on the setting of (8) in Example 2.7, the lifting $\mathsf{Rel}(L)$ is given by

$$\mathsf{inl}(*)\ \ \mathsf{Rel}(L)(\sim)\ \ \mathsf{inr}(*) \tag{24}$$

$$j((a,x))\ \ \mathsf{Rel}(L)(\sim)\ \ j'((b,x)) \iff a = b \text{ and } j(x) \sim j'(y) \tag{25}$$

If $f\colon X \leftarrow 1 + \Sigma \times X$ is an L-coalgebra, we see that $f^* \circ \mathsf{Rel}(L)_X$ maps an equivalence relation $\sim$ on $X + X$ to the least equivalence relation $\approx$ satisfying

$$\mathsf{inl}(f(*)) \approx \mathsf{inr}(f(*)) \tag{26}$$

$$j(f(a,x)) \approx j'(f(a,y)) \text{ whenever } j(x) \sim j'(y) \tag{27}$$

A post-fixed point of this map is an equivalence relation $\sim$ which relates $\mathsf{inl}(f(*))$ and $\mathsf{inr}(f(*))$ and is closed under the action of Σ on $X + X$. The greatest post-fixed point is the *least* such relation, as relations in Rel_X are ordered by reverse inclusion. It is easy to see that this is exactly the relation which identifies $\mathsf{inl}(x)$ and $\mathsf{inr}(x)$ for those x reachable from $f(*)$.

$\mathsf{Rel}(Q)$, meanwhile, maps an equivalence relation $\sim$ on $X + X$ to the relation R on 2^X given by

$$uRv \iff \mathsf{inl}[u] \cup \mathsf{inr}[v] \text{ is } {\sim}\text{-closed} \tag{28}$$

If X' is the set of reachable states, we conclude that $\mathsf{Rel}(Q)$ maps the greatest bisimulation $\sim$ to the relation

$$uRv \iff u \cap X' = v \cap X' \tag{29}$$

The functor Q preserves (abstract) epis, as all epis in $\mathsf{Set}^{\mathsf{op}}$ are regular. Now, Theorem 4.10 tells us that the relation (29) coincides with bisimilarity on the automaton $\overline{Q}(X, f)$ from Example 2.7. It follows that the subautomaton on $2^{X'}$ is minimal, and is the minimal automaton equivalent to $\overline{Q}(X, f)$.

6 Discussion and Future Work

We studied the notion of an *invertible step*, which provides several constructions on coalgebras via functor liftings. We showed that the lifting of the right adjoint, induced by such an invertible step, preserves and reflects bisimilarity. This abstract result instantiates to several concrete results from the literature, in examples related to ultrafilter extensions and weak distributive laws.

We have focused on preservation and reflection of bisimilarity, defined in terms of relation lifting. There are several other coalgebraic notions of behavioural equivalence and bisimilarity [44]—we discuss these in the next subsection. Finally, in Section 6.2 we list directions for future work.

6.1 Remarks on other notions of bisimulation

Aczel-Mendler bisimulations For a coalgebra $f\colon X \to LX$, an Aczel-Mendler bisimulation $R \rightarrowtail X \times X$ is defined by the existence of an L-coalgebra structure $R \to LR$ on R such that the projection maps are coalgebra homomorphisms [1].

In the invertible step setting, applying a lifting $\overline{Q}$ to such a bisimulation, yields a structure $QR \to BQR$. However, this is not immediately a bisimulation, as QR may not be a relation. We can obtain a relation by taking the image of $\langle Q\pi_1, Q\pi_2\rangle$ as we do to define relation lifting, but in general this is a Hermida-Jacobs bisimulation [28, Exercise 4.5.2], rather than an Aczel-Mendler one.

On the other hand, if we wish to speak of reflection of Aczel-Mendler bisimulations, we start with a span $QX \leftarrow R \to QX$ and try to construct a relation on X. Using the adjunction of the step setting, we can transpose the projections to obtain a span $X \leftarrow PR \to X$. Again PR is not immediately a relation in general, and taking the image yields a $\mathsf{Rel}(L)$-coalgebra (not an L-coalgebra) as the projections and the counit ε are coalgebra homomorphisms (see also [28, Exercise 4.5.4]). This in fact comes down to the same as the left adjoint $\coprod_\varepsilon \circ \mathsf{Rel}(P)_{QX}$ constructed earlier. There we factorise to obtain the relation lifting and factorise again for the direct image of ε, instead of factorising the paired transposes defined using ε. We also do not explicitly use that ε is a coalgebra homomorphism (although this follows from the step with right inverse and Lemma 2.3); instead we lift the adjunction at the level of relations to give a map between bisimulations. This is part of the motivation for the use of relation liftings and the corresponding notion of bisimulations.

Going further, it is shown in [5] that there exists a Vietoris bisimulation which is not an Aczel-Mendler bisimulation and, stronger, that there exist Vietoris coalgebras with states which can be related by a Vietoris but not an Aczel-Mendler bisimulation. Thus, the correspondences between bisimulations on Set and Stone we have discussed in the previous sections are not obtainable when we consider Aczel-Mendler bisimulations.

Kernel bisimulations/behavioural equivalence In applying our results to the preservation and reflection of behavioural equivalence, we currently work concretely; considering sets of states and identification of elements.

We prefer to work more abstractly, as we have done for bisimilarity. To this end, we may consider kernel bisimulations. A relation $R \rightarrowtail X \times X$ is a kernel bisimulation on a coalgebra $(X, f\colon X \to LX)$ in a category $\mathcal{D}$, if it is the pullback of morphisms $X \to Z \leftarrow X$ in $\mathcal{D}$ forming a cospan of coalgebra homomorphisms $(X, f) \to (Z, z) \leftarrow (X, f)$ in $\mathsf{Coalg}_{\mathcal{D}}(L)$. In a concrete setting this coincides with behavioural equivalence, as such a pullback contains exactly the pairs of elements of X which are identified in Z by the morphisms forming the cospan. We can thus view this as a generalisation of behavioural equivalence as defined earlier.

Assuming an invertible step $\delta\colon BQ \to QL$, we would like to relate R to a kernel bisimulation on the coalgebra $\overline{Q}(X, f)$ obtained by applying the step-induced lifting of Q. Applying Q to the pullback square for R yields a pullback square as Q is a right adjoint. However, as in our discussion of Aczel-Mendler bisimulations, this may not be a relation. We may try to also use relation liftings here, and take $\mathsf{Rel}(Q)(R)$ instead of $Q(R)$, however this may no longer be a pullback. It is not currently clear to us how to resolve these problems in general.

6.2 Future work

There are several further directions for future work. First, in this paper we focused primarily on fibrations of *relations*, which suffice for our purposes of studying bisimilarity. However, we expect that some of our results can be generalised to arbitrary (posetal) fibrations. Such a generalisation could be the basis to study preservation and reflection of other coinductive predicates and relations than bisimilarity, which can be formulated in terms of fibrations and liftings (e.g., [25]).

Secondly, while we have shown in Section 5 how our results can be used to recover the central results from [5], the latter have been generalised in two directions: the recent [24] considers bisimulations for Vietoris coalgebras on the category of *arbitrary* topological spaces, while [18] develops a notion of neighbourhood bisimulation for coalgebras that allows to generalise the results from [5] to a large variety of functors on the category of Stone spaces and their corresponding functors on Set. We would like to understand whether or not our framework is able to recover these generalisations.

Finally, the examples that we have studied in this paper do not yet exploit the full generality of invertible steps: our main motivating examples are based on an Eilenberg-Moore adjunction (or close, as in the example based on Stone spaces). In [41] it is shown that steps are relevant in a much wider setting, for instance when based on a Kleisli adjunction or on contravariant adjunctions and dualities. The latter type of steps are relevant for coalgebraic modal logics—we have studied a first instance in our example of deterministic and non-deterministic automata. Investigating the meaning of invertible steps in these other types of adjunctions is left for future work.

Acknowledgements This research has been partially funded by the NWO grant OCENW.M20.053 and by Leverhulme Trust Research Project Grant RPG-2020-232.

References

1. Aczel, P., Mendler, N.P.: A final coalgebra theorem. In: Category Theory and Computer Science. Lecture Notes in Computer Science, vol. 389, pp. 357–365. Springer (1989)
2. Adámek, J., Herrlich, H., Strecker, G.E.: Abstract and Concrete Categories - The Joy of Cats. Dover Publications (2009)
3. Bartels, F., Sokolova, A., de Vink, E.P.: A hierarchy of probabilistic system types. Theor. Comput. Sci. **327**(1-2), 3–22 (2004)
4. van Benthem, J.: Canonical modal logics and ultrafilter extensions. The Journal of Symbolic Logic **44**(1), 1–8 (1979), publisher: Cambridge University Press
5. Bezhanishvili, N., Fontaine, G., Venema, Y.: Vietoris bisimulations. J. Log. Comput. **20**(5), 1017–1040 (2010)
6. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic, Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press (2001)
7. Bonchi, F., Bonsangue, M.M., Boreale, M., Rutten, J.J.M.M., Silva, A.: A coalgebraic perspective on linear weighted automata. Inf. Comput. **211**, 77–105 (2012)
8. Bonchi, F., Bonsangue, M.M., Caltais, G., Rutten, J., Silva, A.: A coalgebraic view on decorated traces. Math. Struct. Comput. Sci. **26**(7), 1234–1268 (2016)
9. Bonchi, F., Petrisan, D., Pous, D., Rot, J.: A general account of coinduction up-to. Acta Informatica **54**(2), 127–190 (2017)
10. Bonchi, F., Santamaria, A.: Combining semilattices and semimodules. In: FoSSaCS. Lecture Notes in Computer Science, vol. 12650, pp. 102–123. Springer (2021)
11. Bonchi, F., Silva, A., Sokolova, A.: The power of convex algebras. In: CONCUR. LIPIcs, vol. 85, pp. 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
12. Bonchi, F., Silva, A., Sokolova, A.: Distribution bisimilarity via the power of convex algebras. Log. Methods Comput. Sci. **17**(3) (2021)
13. Bonsangue, M.M., Hansen, H.H., Kurz, A., Rot, J.: Presenting distributive laws. Log. Methods Comput. Sci. **11**(3) (2015)
14. Bonsangue, M.M., Kurz, A.: Duality for logics of transition systems. In: FoSSaCS. Lecture Notes in Computer Science, vol. 3441, pp. 455–469. Springer (2005)
15. Bonsangue, M.M., Milius, S., Silva, A.: Sound and complete axiomatizations of coalgebraic language equivalence. ACM Trans. Comput. Log. **14**(1), 7:1–7:52 (2013)
16. Borceux, F.: Handbook of categorical algebra: volume 1, Basic category theory, vol. 1. Cambridge University Press (1994)
17. Chen, L., Jung, A.: On a categorical framework for coalgebraic modal logic. In: MFPS. Electronic Notes in Theoretical Computer Science, vol. 308, pp. 109–128. Elsevier (2014)
18. Enqvist, S., Sourabh, S.: Bisimulations for coalgebras on Stone spaces. J. Log. Comput. **28**(6), 991–1010 (2018)
19. Garner, R.: The Vietoris monad and weak distributive laws. Appl. Categorical Struct. **28**(2), 339–354 (2020)
20. Goldblatt, R.I.: Metamathematics of modal logic. Bulletin of the Australian Mathematical Society **10**(3), 479–480 (1974), publisher: Cambridge University Press
21. Goy, A.: On the compositionality of monads via weak distributive laws. (Compositionnalité des monades par lois de distributivité faibles). Ph.D. thesis, University of Paris-Saclay, France (2021)
22. Goy, A., Petrisan, D.: Combining probabilistic and non-deterministic choice via weak distributive laws. In: LICS. pp. 454–464. ACM (2020)

23. Goy, A., Petrisan, D., Aiguier, M.: Powerset-like monads weakly distribute over themselves in toposes and compact Hausdorff spaces. In: ICALP. LIPIcs, vol. 198, pp. 132:1–132:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
24. Gumm, H.P., Taheri, M.: Saturated Kripke structures as Vietoris coalgebras. In: CMCS. Lecture Notes in Computer Science, vol. 13225, pp. 88–109. Springer (2022)
25. Hasuo, I., Kataoka, T., Cho, K.: Coinductive predicates and final sequences in a fibration. Math. Struct. Comput. Sci. **28**(4), 562–611 (2018)
26. Hermida, C.: On fibred adjunctions and completeness for fibred categories. In: COMPASS/ADT. Lecture Notes in Computer Science, vol. 785, pp. 235–251. Springer (1992)
27. Hermida, C., Jacobs, B.: Structural induction and coinduction in a fibrational setting. Inf. Comput. **145**(2), 107–152 (1998)
28. Jacobs, B.: Introduction to Coalgebra: Towards Mathematics of States and Observation, Cambridge Tracts in Theoretical Computer Science, vol. 59. Cambridge University Press (2016)
29. Jacobs, B., Silva, A., Sokolova, A.: Trace semantics via determinization. J. Comput. Syst. Sci. **81**(5), 859–879 (2015)
30. Jacobs, B.P.F.: Categorical Logic and Type Theory, Studies in logic and the foundations of mathematics, vol. 141. North-Holland (2001)
31. Kelly, G.M., Street, R.: Review of the elements of 2-categories. In: Kelly, G.M. (ed.) Category Seminar: Proceedings Sydney Category Seminar 1972/1973. No. 420 in Lecture Notes in Mathematics, Springer-Verlag (1974)
32. Klin, B.: Coalgebraic modal logic beyond sets. In: MFPS. Electronic Notes in Theoretical Computer Science, vol. 173, pp. 177–201. Elsevier (2007)
33. Klin, B.: Bialgebras for structural operational semantics: An introduction. Theor. Comput. Sci. **412**(38), 5043–5069 (2011)
34. Klin, B., Salamanca, J.: Iterated covariant powerset is not a monad. In: MFPS. Electronic Notes in Theoretical Computer Science, vol. 341, pp. 261–276. Elsevier (2018)
35. Kupke, C., Kurz, A., Pattinson, D.: Algebraic semantics for coalgebraic logics. In: CMCS. Electronic Notes in Theoretical Computer Science, vol. 106, pp. 219–241. Elsevier (2004)
36. Kupke, C., Kurz, A., Pattinson, D.: Ultrafilter extensions for coalgebras. In: CALCO. Lecture Notes in Computer Science, vol. 3629, pp. 263–277. Springer (2005)
37. Leinster, T.: Higher Operads, Higher Categories, London Mathematical Society Lecture Notes, vol. 298. Cambridge University Press (2004)
38. Levy, P.B.: Final coalgebras from corecursive algebras. In: CALCO. LIPIcs, vol. 35, pp. 221–237. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015)
39. Manes, E.: A triple theoretic construction of compact algebras. In: Seminar on triples and categorical homology theory. pp. 91–118. Springer (1969)
40. Pavlovic, D., Mislove, M.W., Worrell, J.: Testing semantics: Connecting processes and process logics. In: AMAST. Lecture Notes in Computer Science, vol. 4019, pp. 308–322. Springer (2006)
41. Rot, J., Jacobs, B., Levy, P.B.: Steps and traces. J. Log. Comput. **31**(6), 1482–1525 (2021)
42. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theor. Comput. Sci. **249**(1), 3–80 (2000)
43. Silva, A., Bonchi, F., Bonsangue, M.M., Rutten, J.J.M.M.: Generalizing determinization from automata to coalgebras. Log. Methods Comput. Sci. **9**(1) (2013)
44. Staton, S.: Relating coalgebraic notions of bisimulation. Log. Methods Comput. Sci. **7**(1) (2011)

45. Turi, D., Plotkin, G.D.: Towards a mathematical operational semantics. In: LICS. pp. 280–291. IEEE Computer Society (1997)
46. Varacca, D.: Probability, Nondeterminism and Concurrency: Two Denotational Models for Probabilistic Computation. Ph.D. thesis, University of Aarhus (2003)
47. Zwart, M., Marsden, D.: No-go theorems for distributive laws. Log. Methods Comput. Sci. **18**(1) (2022)

Quantitative Safety and Liveness

Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç(✉)

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria
{tah,nmazzocc,esarac}@ist.ac.at

Abstract. Safety and liveness are elementary concepts of computation, and the foundation of many verification paradigms. The safety-liveness classification of boolean properties characterizes whether a given property can be falsified by observing a finite prefix of an infinite computation trace (always for safety, never for liveness). In quantitative specification and verification, properties assign not truth values, but quantitative values to infinite traces (e.g., a cost, or the distance to a boolean property). We introduce quantitative safety and liveness, and we prove that our definitions induce conservative quantitative generalizations of both (1) the safety-progress hierarchy of boolean properties and (2) the safety-liveness decomposition of boolean properties. In particular, we show that every quantitative property can be written as the pointwise minimum of a quantitative safety property and a quantitative liveness property. Consequently, like boolean properties, also quantitative properties can be min-decomposed into safety and liveness parts, or alternatively, max-decomposed into co-safety and co-liveness parts. Moreover, quantitative properties can be approximated naturally. We prove that every quantitative property that has both safe and co-safe approximations can be monitored arbitrarily precisely by a monitor that uses only a finite number of states.

1 Introduction

Safety and liveness are elementary concepts in the semantics of computation [39]. They can be explained through the thought experiment of a *ghost monitor*—an imaginary device that watches an infinite computation trace at runtime, one observation at a time, and always maintains the set of *possible prediction values* to reflect the satisfaction of a given property. Let Φ be a boolean property, meaning that Φ divides all infinite traces into those that satisfy Φ, and those that violate Φ. After any finite number of observations, `True` is a possible prediction value for Φ if the observations seen so far are consistent with an infinite trace that satisfies Φ, and `False` is a possible prediction value for Φ if the observations seen so far are consistent with an infinite trace that violates Φ. When `True` is no possible prediction value, the ghost monitor can reject the hypothesis that Φ is satisfied. The property Φ is *safe* if and only if the ghost monitor can always reject the hypothesis Φ after a finite number of observations: if the infinite trace that is being monitored violates Φ, then after some finite number of observations, `True` is no possible prediction value for Φ. Orthogonally, the property Φ is *live* if and only if the ghost monitor can never reject the hypothesis Φ after a finite number of

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 349–370, 2023.
https://doi.org/10.1007/978-3-031-30829-1_17

observations: for all infinite traces, after every finite number of observations, `True` remains a possible prediction value for Φ.

The safety-liveness classification of properties is fundamental in verification. In the natural topology on infinite traces—the "Cantor topology"—the safety properties are the closed sets, and the liveness properties are the dense sets [4]. For every property Φ, the location of Φ within the Borel hierarchy that is induced by the Cantor topology—the so-called "safety-progress hierarchy" [17]—indicates the level of difficulty encountered when verifying Φ. On the first level, we find the safety and co-safety properties, the latter being the complements of safety properties, i.e., the properties whose falsehood (rather than truth) can always be rejected after a finite number of observations by the ghost monitor. More sophisticated verification techniques are needed for second-level properties, which are the countable boolean combinations of first-level properties—the so-called "response" and "persistence" properties [17]. Moreover, the orthogonality of safety and liveness leads to the following celebrated fact: *every* property can be written as the intersection of a safety property and a liveness property [4]. This means that every property Φ can be decomposed into two parts: a safety part—which is amenable to simple verification techniques, such as invariants—and a liveness part—which requires heavier verification paradigms, such as ranking functions. Dually, there is always a disjunctive decomposition of Φ into co-safety and co-liveness.

So far, we have retold the well-known story of safety and liveness for *boolean* properties. A boolean property Φ is formalized mathematically as the *set* of infinite computation traces that satisfy Φ, or equivalently, the characteristic *function* that maps each infinite trace to a truth value. Quantitative generalizations of the boolean setting allow us to capture not only correctness properties, but also performance properties [31]. In this paper we reveal the story of safety and liveness for such *quantitative* properties, which are functions from infinite traces to an arbitrary set $\mathbb{D}$ of *values*. In order to compare values, we equip the value domain $\mathbb{D}$ with a partial order $<$, and we require $(\mathbb{D}, <)$ to be a complete lattice. The membership problem [18] for an infinite trace f and a quantitative property Φ asks whether $\Phi(f) \geq v$ for a given threshold value $v \in \mathbb{D}$. Correspondingly, in our thought experiment, the ghost monitor attempts to reject hypotheses of the form $\Phi(f) \geq v$, which cannot be rejected as long as all observations seen so far are consistent with an infinite trace f with $\Phi(f) \geq v$. We will define Φ to be a *quantitative safety* property if and only if every hypothesis of the form $\Phi(f) \geq v$ can always be rejected by the ghost monitor after a finite number of observations, and we will define Φ to be a *quantitative liveness* property if and only if some hypothesis of the form $\Phi(f) \geq v$ can never be rejected by the ghost monitor after any finite number of observations. We note that in the quantitative case, after every finite number of observations, the set of possible prediction values for Φ maintained by the ghost monitor may be finite or infinite, and in the latter case, it may not contain a minimal or maximal element.

Let us give a few examples. Suppose we have four observations: observation `rq` for "request a resource," observation `gr` for "grant the resource," observation `tk` for "clock tick," and observation `oo` for "other." The boolean property

Resp requires that every occurrence of `rq` in an infinite trace is followed eventually by an occurrence of `gr`. The boolean property NoDoubleReq requires that no occurrence of `rq` is followed by another `rq` without some `gr` in between. The quantitative property MinRespTime maps every infinite trace to the largest number k such that there are at least k occurrences of `tk` between each `rq` and the closest subsequent `gr`. The quantitative property MaxRespTime maps every infinite trace to the smallest number k such that there are at most k occurrences of `tk` between each `rq` and the closest subsequent `gr`. The quantitative property AvgRespTime maps every infinite trace to the lower limit value $\liminf$ of the infinite sequence $(v_i)_{i \geq 1}$, where v_i is, for the first i occurrences of `tk`, the average number of occurrences of `tk` between `rq` and the closest subsequent `gr`. Note that the values of AvgRespTime can be ∞ for some computations, including those for which the value of Resp is `True`. This highlights that boolean properties are not embedded in the limit behavior of quantitative properties.

The boolean property Resp is live because every finite observation sequence can be extended with an occurrence of `gr`. In fact, Resp is a second-level liveness property (namely, a response property), because it can be written as a countable intersection of co-safety properties. The boolean property NoDoubleReq is safe because if it is violated, it will be rejected by the ghost monitor after a finite number of observations, namely, as soon as the ghost monitor sees a `rq` followed by another occurrence of `rq` without an intervening `gr`. According to our quantitative generalization of safety, MinRespTime is a safety property. The ghost monitor always maintains the minimal number k of occurrences of `tk` between any past `rq` and the closest subsequent `gr` seen so far; the set of possible prediction values for MinRespTime is always $\{0, 1, \ldots, k\}$. Every hypothesis of the form "the MinRespTime-value is at least v" is rejected by the ghost monitor as soon as $k < v$; if such a hypothesis is violated, this will happen after some finite number of observations. Symmetrically, the quantitative property MaxRespTime is co-safe, because every wrong hypothesis of the form "the MaxRespTime-value is at most v" will be rejected by the ghost monitor as soon as the smallest possible prediction value for MaxRespTime, which is the maximal number of occurrences of `tk` between any past `rq` and the closest subsequent `gr` seen so far, goes above v. By contrast, the quantitative property AvgRespTime is both live and co-live because no hypothesis of the form "the AvgRespTime-value is at least v," nor of the form "the AvgRespTime-value is at most v," can ever be rejected by the ghost monitor after a finite number of observations. All nonnegative real numbers and ∞ always remain possible prediction values for AvgRespTime. Note that a ghost monitor that attempts to reject hypotheses of the form $\Phi(f) \geq v$ does not need to maintain the entire set of possible prediction values, but only the sup of the set of possible prediction values, and whether or not the sup is contained in the set. Dually, updating inf (and whether it is contained) suffices to reject hypotheses of the form $\Phi(f) \leq v$.

By defining quantitative safety and liveness via ghost monitors, we not only obtain a conservative and quantitative generalization of the boolean story, but also open up attractive frontiers for quantitative semantics, monitoring, and verification. For example, while the approximation of boolean properties reduces to

adding and removing traces to and from a set, the approximation of quantitative properties offers a rich landscape of possibilities. In fact, we can approximate the notion of safety itself. Given an error bound α, the quantitative property Φ is α-*safe* if and only if for every value v and every infinite trace f whose value $\Phi(f)$ is less than v, all possible prediction values for Φ are less than $v + \alpha$ after some finite prefix of f. This means that, for an α-safe property Φ, the ghost monitor may not reject wrong hypotheses of the form $\Phi(f) \geq v$ after a finite number of observations, once the violation is below the error bound. We show that every quantitative property that is both α-safe and β-co-safe, for any finite α and β, can be monitored arbitrarily precisely by a monitor that uses only a finite number of states.

We are not the first to define quantitative (or multi-valued) definitions of safety and liveness [41,27]. While the previously proposed quantitative generalizations of safety share strong similarities with our definition (without coinciding completely), our quantitative generalization of liveness is entirely new. The definitions of [27] do not support any safety-liveness decomposition, because their notion of safety is too permissive, and their liveness too restrictive. While the definitions of [41] admit a safety-liveness decomposition, our definition of liveness captures strictly fewer properties. Consequently, our definitions offer a stronger safety-liveness decomposition theorem. Our definitions also fit naturally with the definitions of emptiness, equivalence, and inclusion for quantitative languages [18].

Overview. In Section 2, we introduce quantitative properties. In Section 3, we define quantitative safety as well as safety closure, namely, the property that increases the value of each trace as little as possible to achieve safety. Then, we prove that our definitions preserve classical boolean facts. In particular, we show that a quantitative property Φ is safe if and only if Φ equals its safety closure if and only if Φ is upper semicontinuous. In Section 4, we generalize the safety-progress hierarchy to quantitative properties. We first define limit properties. For $\ell \in \{\inf, \sup, \liminf, \limsup\}$, the class of ℓ-properties captures those for which the value of each infinite trace can be derived by applying the limit function ℓ to the infinite sequence of values of finite prefixes. We prove that inf-properties coincide with safety, sup-properties with co-safety, $\liminf$-properties are suprema of countably many safety properties, and $\limsup$-properties infima of countably many co-safety properties. The $\liminf$-properties generalize the boolean persistence properties of [17]; the $\limsup$-properties generalize their response properties. For example, AvgRespTime is a $\liminf$-property. In Section 5, we introduce quantitative liveness and co-liveness. We prove that our definitions preserve the classical boolean facts, and show that there is a unique property which is both safe and live. As main result, we provide a safety-liveness decomposition that holds for every quantitative property. In Section 6, we define approximate safety and co-safety. We generalize the well-known unfolding approximation of discounted properties for approximate safety and co-safety properties over the extended reals. This allows us to provide a finite-state approximate monitor for these properties. In Section 7, we conclude with future research directions. For complete proofs of all results, we refer the reader to the full version of the paper.

Related Work. The notions of safety and liveness for boolean properties appeared first in [39] and were later formalized in [4], where safety properties were characterized as closed sets of the Cantor topology on infinite traces, and liveness properties as dense sets. As a consequence, the seminal decomposition theorem followed: every boolean property is an intersection of a safety property and a liveness property. A benefit of such a decomposition lies in the difference between the mathematical arguments used in their verification. While safety properties enable simpler methods such as invariants, liveness properties require more complex approaches such as well-foundedness [42,5]. These classes were characterized in terms of Büchi automata in [5] and in terms of linear temporal logic in [46].

The safety-progress classification of boolean properties [17] proposes an orthogonal view: rather than partitioning the set of properties, it provides a hierarchy of properties starting from safety. This yields a more fine-grained view of nonsafety properties which distinguishes whether a "good thing" happens at least once (co-safety or "guarantee"), infinitely many times (response), or eventually always (persistence). This classification follows the Borel hierarchy that is induced by the Cantor topology on infinite traces, and has corresponding projections within properties that are definable by finite automata and by formulas of linear temporal logic.

Runtime verification, or monitoring, is a lightweight, dynamic verification technique [6], where a monitor watches a system during its execution and tries to decide, after each finite sequence of observations, whether the observed finite computation trace or its unknown infinite extension satisfies a desired property. The safety-liveness dichotomy has profound implications for runtime verification as well: safety is easy to monitor [28], while liveness is not. An early definition of boolean monitorability was equivalent to safety with recursively enumerable sets of bad prefixes [35]. The monitoring of infinite-state boolean safety properties was later studied in [26]. A more popular definition of boolean monitorability [44,8] accounts for both truth and falsehood, establishing the set of monitorable properties as a strict superset of finite boolean combinations of safety and co-safety [23]. Boolean monitors that use the set possible prediction values can be found in [7]. The notion of boolean monitorability was investigated through the safety-liveness lens in [43] and through the safety-progress lens in [23].

Quantitative properties (a.k.a. "quantitative languages") [18] extend their boolean counterparts by moving from the two-valued truth domain to richer domains such as real numbers. Such properties have been extensively studied from a static verification perspective in the past decade, e.g., in the context of model-checking probabilistic properties [38,37], games with quantitative objectives [10,15], specifying quantitative properties [11,1], measuring distances between systems [2,16,22,29], best-effort synthesis and repair [9,20], and quantitative analysis of transition systems [47,14,21,19]. More recently, quantitative properties have been also studied from a runtime verification perspective, e.g., for limit monitoring of statistical indicators of infinite traces [25] and for analyzing resource-precision trade-offs in the design of quantitative monitors [33,30].

To the best of our knowledge, previous definitions of (approximate) safety and liveness in nonboolean domains make implicit assumptions about the spec-

ification language [48,34,24,45]. We identify two notable exceptions. In [27], the authors generalize the framework of [43] to nonboolean value domains. They provide neither a safety-liveness decomposition of quantitative properties, nor a fine-grained classification of nonsafety properties. In [41], the authors present a safety-liveness decomposition and some levels of the safety-progress hierarchy on multi-valued truth domains, which are bounded distributive lattices. Their motivation is to provide algorithms for model-checking properties on multi-valued truth domains. We present the relationships between their definitions and ours in the relevant sections below.

2 Quantitative Properties

Let $\Sigma = \{a, b, \ldots\}$ be a finite alphabet of observations. A *trace* is an infinite sequence of observations, denoted by $f, g, h \in \Sigma^\omega$, and a *finite trace* is a finite sequence of observations, denoted by $s, r, t \in \Sigma^*$. Given $s \in \Sigma^*$ and $w \in \Sigma^* \cup \Sigma^\omega$, we denote by $s \prec w$ (resp. $s \preceq w$) that s is a strict (resp. nonstrict) prefix of w. Furthermore, we denote by $|w|$ the length of w and, given $a \in \Sigma$, by $|w|_a$ the number of occurrences of a in w.

A *value domain* $\mathbb{D}$ is a poset. Unless otherwise stated, we assume that $\mathbb{D}$ is a nontrivial (i.e., $\bot \neq \top$) complete lattice and, whenever appropriate, we write $0, 1, -\infty, \infty$ instead of $\bot$ and $\top$ for the least and the greatest elements. We respectively use the terms minimum and maximum for the greatest lower bound and the least upper bound of finitely many elements.

Definition 1 (Property). *A* quantitative property *(or simply* property*) is a function $\Phi : \Sigma^\omega \to \mathbb{D}$ from the set of all traces to a value domain.*

A boolean property $P \subseteq \Sigma^\omega$ is defined as a set of traces. We use the boolean domain $\mathbb{B} = \{0, 1\}$ with $0 < 1$ and, in place of P, its *characteristic property* $\Phi_P : \Sigma^\omega \to \mathbb{B}$, which is defined by $\Phi_P(f) = 1$ if $f \in P$, and $\Phi_P(f) = 0$ if $f \notin P$.

For all properties Φ_1, Φ_2 on a domain $\mathbb{D}$ and all traces $f \in \Sigma^\omega$, we let $\min(\Phi_1, \Phi_2)(f) = \min(\Phi_1(f), \Phi_2(f))$ and $\max(\Phi_1, \Phi_2)(f) = \max(\Phi_1(f), \Phi_2(f))$. For a domain $\mathbb{D}$, the *inverse* of $\mathbb{D}$ is the domain $\overline{\mathbb{D}}$ that contains the same elements as $\mathbb{D}$ but with the ordering reversed. For a property Φ, we define its *complement* $\overline{\Phi} : \Sigma^\omega \to \overline{\mathbb{D}}$ by $\overline{\Phi}(f) = \Phi(f)$ for all $f \in \Sigma^\omega$.

Some properties can be defined as limits of value sequences. A *finitary property* $\pi \colon \Sigma^* \to \mathbb{D}$ associates a value with each finite trace. A *value function* $\ell \colon \mathbb{D}^\omega \to \mathbb{D}$ condenses an infinite sequence of values to a single value. Given a finitary property π, a value function ℓ, and a trace $f \in \Sigma^\omega$, we write $\ell_{s \prec f} \pi(s)$ instead of $\ell(\pi(s_0)\pi(s_1)\ldots)$, where each s_i fulfills $s_i \prec f$ and $|s_i| = i$.

3 Quantitative Safety

Given a property $\Phi : \Sigma^\omega \to \mathbb{D}$, a trace $f \in \Sigma^\omega$, and a value $v \in \mathbb{D}$, the quantitative membership problem [18] asks whether $\Phi(f) \geq v$. We define quantitative safety as follows: the property Φ is safe iff every wrong hypothesis of the form $\Phi(f) \geq v$ has a finite witness $s \prec f$.

Definition 2 (Safety). *A property* $\Phi : \Sigma^\omega \to \mathbb{D}$ *is* safe *iff for every* $f \in \Sigma^\omega$ *and value* $v \in \mathbb{D}$ *with* $\Phi(f) \not\geq v$*, there is a prefix* $s \prec f$ *such that* $\sup_{g \in \Sigma^\omega} \Phi(sg) \not\geq v$.

Let us illustrate this definition with the *minimal response-time* property.

Example 3. Let $\Sigma = \{\mathtt{rq}, \mathtt{gr}, \mathtt{tk}, \mathtt{oo}\}$ and $\mathbb{D} = \mathbb{N} \cup \{\infty\}$. We define the minimal response-time property $\Phi_{\min}$ through an auxiliary finitary property $\pi_{\min}$ that computes the minimum response time so far. In a finite or infinite trace, an occurrence of `rq` is *granted* if it is followed, later, by a `gr`, and otherwise it is *pending*. Let $\pi_{\text{last}}(s) = \infty$ if the finite trace s contains a pending `rq`, or no `rq`, and $\pi_{\text{last}}(s) = |r|_{\mathtt{tk}} - |t|_{\mathtt{tk}}$ otherwise, where $r \prec s$ is the longest prefix of s with a pending `rq`, and $t \prec r$ is the longest prefix of r without pending `rq`. Intuitively, π_{last} provides the response time for the last request when all requests are granted, and ∞ when there is a pending request or no request. Given $s \in \Sigma^*$, taking the minimum of the values of π_{last} over the prefixes $r \preceq s$ gives us the minimum response time so far. Let $\pi_{\min}(s) = \min_{r \preceq s} \pi_{\text{last}}(r)$ for all $s \in \Sigma^*$, and $\Phi_{\min}(f) = \lim_{s \prec f} \pi_{\min}(s)$ for all $f \in \Sigma^\omega$. The limit always exists because the minimum is monotonically decreasing.

The minimal response-time property is safe. Let $f \in \Sigma^\omega$ and $v \in \mathbb{D}$ such that $\Phi_{\min}(f) < v$. Then, some prefix $s \prec f$ contains a `rq` that is granted after $u < v$ ticks, in which case, no matter what happens in the future, the minimal response time is guaranteed to be at most u; that is, $\sup_{g \in \Sigma^\omega} \Phi_{\min}(sg) \leq u < v$. If you recall from the introduction the ghost monitor that maintains the sup of possible prediction values for the minimal response-time property, that value is always $\pi_{\min}$; that is, $\sup_{g \in \Sigma^\omega} \Phi_{\min}(sg) = \pi_{\min}(s)$ for all $s \in \Sigma^*$. Note that in the case of minimal response time, the sup of possible prediction values is always realizable; that is, for all $s \in \Sigma^*$, there exists an $f \in \Sigma^\omega$ such that $\sup_{g \in \Sigma^\omega} \Phi_{\min}(sg) = \Phi_{\min}(sf)$. □

Remark 4. Quantitative safety generalizes boolean safety. For every boolean property $P \subseteq \Sigma^\omega$, the following statements are equivalent: (i) P is safe according to the classical definition [4], (ii) its characteristic property Φ_P is safe, and (iii) for every $f \in \Sigma^\omega$ and $v \in \mathbb{B}$ with $\Phi_P(f) < v$, there exists a prefix $s \prec f$ such that for all $g \in \Sigma^\omega$, we have $\Phi_P(sg) < v$.

We now generalize the notion of safety closure and present an operation that makes a property safe by increasing the value of each trace as little as possible.

Definition 5 (Safety closure). *The* safety closure *of a property* Φ *is the property* Φ^* *defined by* $\Phi^*(f) = \inf_{s \prec f} \sup_{g \in \Sigma^\omega} \Phi(sg)$ *for all* $f \in \Sigma^\omega$.

We can say the following about the safety closure operation.

Proposition 6. *For every property* $\Phi : \Sigma^\omega \to \mathbb{D}$*, the following statements hold.*

1. Φ^* *is safe.*
2. $\Phi^*(f) \geq \Phi(f)$ *for all* $f \in \Sigma^\omega$.
3. $\Phi^*(f) = \Phi^{**}(f)$ *for all* $f \in \Sigma^\omega$.
4. *For every safety property* $\Psi : \Sigma^\omega \to \mathbb{D}$*, if* $\Phi(f) \leq \Psi(f)$ *for all* $f \in \Sigma^\omega$*, then* $\Psi(g) \not< \Phi^*(g)$ *for all* $g \in \Sigma^\omega$.

3.1 Alternative Characterizations of Quantitative Safety

Consider a trace and its prefixes of increasing length. For a given property, the ghost monitor from the introduction maintains, for each prefix, the sup of possible prediction values, i.e., the least upper bound of the property values for all possible infinite continuations. The resulting sequence of monotonically decreasing suprema provides an upper bound on the eventual property value. Moreover, for some properties, this sequence always converges to the property value. If this is the case, then the ghost monitor can always dismiss wrong lower-bound hypotheses after finite prefixes, and vice versa. This gives us an alternative definition for the safety of quantitative properties which, inspired by the notion of Scott continuity, was called *continuity* [33]. We now believe that *upper semicontinuity* is a more appropriate term, as becomes clear when we consider the Cantor topology on Σ^ω and the value domain $\mathbb{R} \cup \{-\infty, +\infty\}$.

Definition 7 (Upper semicontinuity [33]). *A property Φ is* upper semicontinuous *iff $\Phi(f) = \lim_{s \prec f} \sup_{g \in \Sigma^\omega} \Phi(sg)$ for all $f \in \Sigma^\omega$.*

We note that the minimal response-time property is upper semicontinuous.

Example 8. Recall the minimal response-time property $\Phi_{\min}$ from Example 3. For every trace $f \in \Sigma^\omega$, the $\Phi_{\min}$ value is the limit of the $\pi_{\min}$ values for the prefixes of f. Therefore, $\Phi_{\min}$ is upper semicontinuous. □

In general, a property is safe iff it maps every trace to the limit of the suprema of possible prediction values. Moreover, we can also characterize safety properties as the properties that are equal to their safety closure.

Theorem 9. *For every property Φ, the following statements are equivalent: 1. Φ is safe. 2. Φ is upper semicontinuous. 3. $\Phi(f) = \Phi^*(f)$ for all $f \in \Sigma^\omega$.*

3.2 Related Definitions of Quantitative Safety

In [41], the authors consider the model-checking problem for properties on multi-valued truth domains. They introduce the notion of multi-safety through a closure operation that coincides with our safety closure. Formally, a property Φ is *multi-safe* iff $\Phi(f) = \Phi^*(f)$ for every $f \in \Sigma^\omega$. It is easy to see the following.

Proposition 10. *For every property Φ, we have Φ is multi-safe iff Φ is safe.*

Although the two definitions of safety are equivalent, our definition is consistent with the membership problem for quantitative automata and motivated by the monitoring of quantitative properties.

In [27], the authors extend a refinement of the safety-liveness classification for monitoring [43] to richer domains. They introduce the notion of verdict-safety through dismissibility of values not less than or equal to the property value. Formally, a property Φ is *verdict-safe* iff for every $f \in \Sigma^\omega$ and $v \not\leq \Phi(f)$, there exists a prefix $s \prec f$ such that for all $g \in \Sigma^\omega$, we have $\Phi(sg) \neq v$.

We demonstrate that verdict-safety is weaker than safety. Moreover, we provide a condition under which the two definitions coincide. To achieve this, we reason about sets of possible prediction values: for a property Φ and $s \in \Sigma^*$, let $P_{\Phi,s} = \{\Phi(sf) \mid f \in \Sigma^\omega\}$.

Lemma 11. *A property Φ is verdict-safe iff $\Phi(f) = \sup(\lim_{s \prec f} P_{\Phi,s})$ for all $f \in \Sigma^\omega$.*

Notice that Φ is safe iff $\Phi(f) = \lim_{s \prec f}(\sup P_{\Phi,s})$ for all $f \in \Sigma^\omega$. Below we describe a property that is verdict-safe but not safe.

Example 12. Let $\Sigma = \{a, b\}$. Define Φ by $\Phi(f) = 0$ if $f = a^\omega$, and $\Phi(f) = |s|$ otherwise, where $s \prec f$ is the shortest prefix in which b occurs. The property Φ is verdict-safe. First, observe that $\mathbb{D} = \mathbb{N} \cup \{\infty\}$. Let $f \in \Sigma^\omega$ and $v \in \mathbb{D}$ with $v > \Phi(f)$. If $\Phi(f) > 0$, then f contains b, and $\Phi(f) = |s|$ for some $s \prec f$ in which b occurs for the first time. After the prefix s, all $g \in \Sigma^\omega$ yield $\Phi(sg) = |s|$, thus all values above $|s|$ are rejected. If $\Phi(f) = 0$, then $f = a^\omega$. Let $v \in \mathbb{D}$ with $v > 0$, and consider the prefix $a^v \prec f$. Observe that the set of possible prediction values after reading a^v is $\{0, v+1, v+2, \ldots\}$, therefore a^v allows the ghost monitor to reject the value v. However, Φ is not safe because, although $\Phi(a^\omega) = 0$, for every $s \prec a^\omega$, we have $\sup_{g \in \Sigma^\omega} \Phi(sg) = \infty$. □

The separation is due to the fact that, for some finite traces, the sup of possible prediction values cannot be realized by any future. Below, we present a condition that prevents such cases.

Definition 13 (Supremum closedness). *A property Φ is* sup-closed *iff for every $s \in \Sigma^*$ we have* $\sup P_{\Phi,s} \in P_{\Phi,s}$.

We remark that the minimal response-time property is sup-closed.

Example 14. The safety property minimal response-time $\Phi_{\min}$ from Example 3 is sup-closed. This is because, for every $s \in \Sigma^*$, the continuation $\mathtt{gr}^\omega$ realizes the value $\sup_{g \in \Sigma^\omega} \Phi(sg)$. □

Recall from the introduction the ghost monitor that maintains the sup of possible prediction values. For monitoring sup-closed properties this suffices; otherwise the ghost monitor also needs to maintain whether or not the supremum of the possible prediction values is realizable by some future continuation. In general, we have the following for every sup-closed property.

Lemma 15. *For every* sup*-closed property Φ and for all $f \in \Sigma^\omega$, we have* $\lim_{s \prec f}(\sup P_{\Phi,s}) = \sup(\lim_{s \prec f} P_{\Phi,s})$.

As a consequence of the lemmas above, we get the following.

Theorem 16. *A* sup*-closed property Φ is safe iff Φ is verdict-safe.*

4 The Quantitative Safety-Progress Hierarchy

Our quantitative extension of safety closure allows us to build a Borel hierarchy, which is a quantitative extension of the boolean safety-progress hierarchy [17]. First, we show that safety properties are closed under pairwise min and max.

Proposition 17. *For every value domain $\mathbb{D}$, the set of safety properties over $\mathbb{D}$ is closed under* min *and* max.

The boolean safety-progress classification of properties is a Borel hierarchy built from the Cantor topology of traces. Safety and co-safety properties lie on the first level, respectively corresponding to the closed sets and open sets of the topology. The second level is obtained through countable unions and intersections of properties from the first level: persistence properties are countable unions of closed sets, while response properties are countable intersections of open sets. We generalize this construction to the quantitative setting.

In the boolean case, each property class is defined through an operation that takes a set $S \subseteq \Sigma^*$ of finite traces and produces a set $P \subseteq \Sigma^\omega$ of infinite traces. For example, to obtain a co-safety property from $S \subseteq \Sigma^*$, the corresponding operation yields $S\Sigma^\omega$. Similarly, we formalize each property class by a value function. For this, we define the notion of *limit property*.

Definition 18 (Limit property). *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is a* limit property *iff there exists a finitary property $\pi : \Sigma^* \to \mathbb{D}$ and a value function $\ell : \mathbb{D}^\omega \to \mathbb{D}$ such that $\Phi(f) = \ell_{s \prec f} \pi(s)$ for all $f \in \Sigma^\omega$. We denote this by $\Phi = (\pi, \ell)$, and write $\Phi(s)$ instead of $\pi(s)$. In particular, if $\Phi = (\pi, \ell)$, where $\ell \in \{\inf, \sup, \liminf, \limsup\}$, then Φ is an ℓ-*property.

To account for the value functions that construct the first two levels of the safety-progress hierarchy, we start our investigation with inf- and sup-properties and later focus on $\liminf$- and $\limsup$- properties [18].

4.1 Infimum and Supremum Properties

Let us start with an example by demonstrating that the minimal response-time property is an inf-property.

Example 19. Recall the safety property $\Phi_{\min}$ of minimal response time from Example 3. We can equivalently define $\Phi_{\min}$ as a limit property by taking the finitary property π_{last} and the value function inf. As discussed in Example 3, the function π_{last} outputs the response time for the last request when all requests are granted, and ∞ when there is a pending request or no request. Then $\inf_{s \prec f} \pi_{\text{last}}(s) = \Phi_{\min}(f)$ for all $f \in \Sigma^\omega$, and therefore $\Phi_{\min} = (\pi_{\text{last}}, \inf)$. □

In fact, the safety properties coincide with inf-properties.

Theorem 20. *A property Φ is safe iff Φ is an* inf*-property.*

Defining the minimal response-time property as a limit property, we observe the following relation between its behavior on finite traces and infinite traces.

Example 21. Consider the property $\Phi_{\min} = (\pi_{\text{last}}, \inf)$ from Example 19. Let $f \in \Sigma^\omega$ and $v \in \mathbb{D}$. Observe that if the minimal response time of f is at least v, then the last response time for each prefix $s \prec f$ is also at least v. Conversely, if the minimal response time of f is below v, then there is a prefix $s \prec f$ for which the last response time is also below v. □

In light of this observation, we provide another characterization of safety properties, explicitly relating the specified behavior of the limit property on finite and infinite traces.

Theorem 22. *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is safe iff Φ is a limit property such that for every $f \in \Sigma^\omega$ and value $v \in \mathbb{D}$, we have $\Phi(f) \geq v$ iff $\Phi(s) \geq v$ for all $s \prec f$.*

Recall that a safety property allows rejecting wrong lower-bound hypotheses with a finite witness, by assigning a tight upper bound to each trace. We define co-safety properties symmetrically: a property Φ is co-safe iff every wrong hypothesis of the form $\Phi(f) \leq v$ has a finite witness $s \prec f$.

Definition 23 (Co-safety). *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* co-safe *iff for every $f \in \Sigma^\omega$ and value $v \in \mathbb{D}$ with $\Phi(f) \not\leq v$, there exists a prefix $s \prec f$ such that $\inf_{g \in \Sigma^\omega} \Phi(sg) \not\leq v$.*

We note that our definition generalizes boolean co-safety, and thus a dual of Remark 4 holds also for co-safety. Moreover, we analogously define the notions of co-safety closure and lower semicontinuity.

Definition 24 (Co-safety closure). *The* co-safety closure *of a property Φ is the property $\Phi_*(f)$ defined by $\Phi_*(f) = \sup_{s \prec f} \inf_{g \in \Sigma^\omega} \Phi(sg)$ for all $f \in \Sigma^\omega$.*

Definition 25 (Lower semicontinuity [33]). *A property Φ is* lower semicontinuous *iff $\Phi(f) = \lim_{s \prec f} \inf_{g \in \Sigma^\omega} \Phi(sg)$ for all $f \in \Sigma^\omega$.*

Now, we define and investigate the *maximal response-time* property. In particular, we show that it is a sup-property that is co-safe and lower semicontinuous.

Example 26. Let $\Sigma = \{\mathtt{rq}, \mathtt{gr}, \mathtt{tk}, \mathtt{oo}\}$ and $\mathbb{D} = \mathbb{N} \cup \{\infty\}$. We define the maximal response-time property $\Phi_{\max}$ through a finitary property that computes the current response time for each finite trace and the value function sup. In particular, for all $s \in \Sigma^*$, let $\pi_{\text{curr}}(s) = |s|_{\mathtt{tk}} - |r|_{\mathtt{tk}}$, where $r \preceq s$ is the longest prefix of s without pending $\mathtt{rq}$; then $\Phi_{\max} = (\pi_{\text{curr}}, \sup)$. Note the contrast between π_{curr} and π_{last} from Example 3. While π_{curr} takes an optimistic view of the future and assumes the $\mathtt{gr}$ will follow immediately, π_{last} takes a pessimistic view and assumes the $\mathtt{gr}$ will never follow. Let $f \in \Sigma^\omega$ and $v \in \mathbb{D}$. If the maximal response time of f is greater than v, then for some prefix $s \prec f$ the current response time is greater than v also, which means that, no matter what happens in the future, the maximal response time is greater than v after observing s. Therefore, $\Phi_{\max}$ is co-safe. By a similar reasoning, the sequence of greatest lower bounds of possible prediction values over the prefixes converges to the property value. In other words, we have $\lim_{s \prec f} \inf_{g \in \Sigma^\omega} \Phi_{\max}(sg) = \Phi_{\max}(f)$ for all $f \in \Sigma^\omega$. Thus $\Phi_{\max}$ is also lower semicontinuous, and it equals its co-safety closure. Now, consider the complementary property $\overline{\Phi_{\max}}$, which maps every trace to the same value as $\Phi_{\max}$ on a domain where the order is reversed. It is easy to see that $\overline{\Phi_{\max}}$ is safe. Finally, recall the ghost monitor from the introduction, which maintains the infimum of possible prediction values for the maximal response-time property. Since the maximal response-time property is inf-closed, the output of the ghost monitor after every prefix is realizable by some future continuation, and that output is $\pi_{\max}(s) = \max_{r \preceq s} \pi_{\text{curr}}(r)$ for all $s \in \Sigma^*$. □

Generalizing the observations in the example above, we obtain the following characterizations due to the duality between safety and co-safety.

Theorem 27. *For every property $\Phi : \Sigma^\omega \to \mathbb{D}$, the following are equivalent.*

1. *Φ is co-safe.*
2. *Φ is lower semicontinuous.*
3. *$\Phi(f) = \Phi_*(f)$ for every $f \in \Sigma^\omega$.*
4. *Φ is a* sup*-property.*
5. *Φ is a limit property such that for every $f \in \Sigma^\omega$ and value $v \in \mathbb{D}$, we have $\Phi(f) \leq v$ iff $\Phi(s) \leq v$ for all $s \prec f$.*
6. *$\overline{\Phi}$ is safe.*

4.2 Limit Inferior and Limit Superior Properties

Let us start with an observation on the minimal response-time property.

Example 28. Recall once again the minimal response-time property $\Phi_{\min}$ from Example 3. In the previous subsection, we presented an alternative definition of $\Phi_{\min}$ to establish that it is an inf-property. Observe that there is yet another equivalent definition of $\Phi_{\min}$ which takes the monotonically decreasing finitary property $\pi_{\min}$ from Example 3 and pairs it with either the value function $\liminf$, or with $\limsup$. Hence $\Phi_{\min}$ is both a $\liminf$- and a $\limsup$-property. □

Before moving on to investigating $\liminf$- and $\limsup$-properties more closely, we show that the above observation can be generalized.

Theorem 29. *Every ℓ-property Φ, for $\ell \in \{\inf, \sup\}$, is both a* $\liminf$- *and a* $\limsup$*-property.*

An interesting response-time property beyond safety and co-safety arises when we remove extreme values: instead of minimal response time, consider the property that maps every trace to a value that bounds from below, not all response times, but all of them from a point onward (i.e., all but finitely many). We call this property *tail-minimal response time*.

Example 30. Let $\Sigma = \{\mathtt{rq}, \mathtt{gr}, \mathtt{tk}, \mathtt{oo}\}$ and π_{last} be the finitary property from Example 3 that computes the last response time. We define the tail-minimal response-time property as $\Phi_{\text{tmin}} = (\pi_{\text{last}}, \liminf)$. Intuitively, it maps each trace to the least response time over all but finitely many requests. This property is interesting as a performance measure, because it focuses on the long-term performance by ignoring finitely many outliers. Consider $f \in \Sigma^\omega$ and $v \in \mathbb{D}$. Observe that, if the tail-minimal response time of f is at least v, then there is a prefix $s \prec f$ such that for all longer prefixes $s \preceq r \prec f$, the last response time in r is at least v, and vice versa. □

Similarly as for inf-properties, we characterize $\liminf$-properties through a relation between property behaviors on finite and infinite traces.

Theorem 31. *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is a* $\liminf$*-property iff Φ is a limit property such that for every $f \in \Sigma^\omega$ and value $v \in \mathbb{D}$, we have $\Phi(f) \geq v$ iff there exists $s \prec f$ such that for all $s \preceq r \prec f$, we have $\Phi(r) \geq v$.*

Now, we show that the tail-minimal response-time property can be expressed as a countable supremum of inf-properties.

Example 32. Let $i \in \mathbb{N}$ and define $\pi_{i,\text{last}}$ as a finitary property that imitates π_{last} from Example 3, but ignores the first i observations of every finite trace. Formally, for $s \in \Sigma^*$, we define $\pi_{i,\text{last}}(s) = \pi_{\text{last}}(r)$ for $s = s_i r$ where $s_i \preceq s$ with $|s_i| = i$, and $r \in \Sigma^*$. Observe that an equivalent way to define Φ_{tmin} from Example 30 is $\sup_{i\in\mathbb{N}}(\inf_{s\prec f}(\pi_{i,\text{last}}(s)))$ for all $f \in \Sigma^\omega$. Intuitively, for each $i \in \mathbb{N}$, we obtain an inf-property that computes the minimal response time of the suffixes of a given trace. Taking the supremum over these, we obtain the greatest lower bound on all but finitely many response times. □

We generalize this observation and show that every lim inf-property is a countable supremum of inf-properties.

Theorem 33. *Every* lim inf*-property is a countable supremum of* inf*-properties.*

We would also like to have the converse of Theorem 33, i.e., that every countable supremum of inf-properties is a lim inf-property. Currently, we are able to show only the following.

Theorem 34. *For every infinite sequence* $(\Phi_i)_{i\in\mathbb{N}}$ *of* inf*-properties, there is a* lim inf*-property* Φ *such that* $\sup_{i\in\mathbb{N}} \Phi_i(f) \leq \Phi(f)$.

We conjecture that some lim inf-property that satisfies Theorem 34 is also a lower bound on the countable supremum that occurs in the theorem. This, together with Theorem 34, would imply the converse of Theorem 33. Proving the converse of Theorem 33 would give us, thanks to the following duality, that the lim inf- and lim sup-properties characterize the second level of the Borel hierarchy of the topology induced by the safety closure operator.

Proposition 35. *A property* Φ *is a* lim inf*-property iff its complement* $\overline{\Phi}$ *is a* lim sup*-property.*

5 Quantitative Liveness

Similarly as for safety, we take the perspective of the quantitative membership problem to define liveness: a property Φ is live iff, whenever a property value is less than $\top$, there exists a value v for which the wrong hypothesis $\Phi(f) \geq v$ can never be dismissed by any finite witness $s \prec f$.

Definition 36 (Liveness). *A property* $\Phi : \Sigma^\omega \to \mathbb{D}$ *is* live *iff for all* $f \in \Sigma^\omega$*, if* $\Phi(f) < \top$*, then there exists a value* $v \in \mathbb{D}$ *such that* $\Phi(f) \not\geq v$ *and for all prefixes* $s \prec f$*, we have* $\sup_{g\in\Sigma^\omega} \Phi(sg) \geq v$.

An equivalent definition can be given through the safety closure.

Theorem 37. *A property* Φ *is live iff* $\Phi^*(f) > \Phi(f)$ *for every* $f \in \Sigma^\omega$ *with* $\Phi(f) < \top$.

Our definition generalizes boolean liveness. A boolean property $P \subseteq \Sigma^\omega$ is live according to the classical definition [4] iff its characteristic property Φ_P is live according to our definition. Moreover, the intersection of safety and liveness contains only the single degenerate property that always outputs $\top$.

Proposition 38. *A property Φ is safe and live iff $\Phi(f) = \top$ for all $f \in \Sigma^\omega$.*

We define co-liveness symmetrically, and note that the duals of the observations above also hold for co-liveness.

Definition 39 (Co-liveness). *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* co-live *iff for all $f \in \Sigma^\omega$, if $\Phi(f) > \bot$, then there exists a value $v \in \mathbb{D}$ such that $\Phi(f) \not\leq v$ and for all prefixes $s \prec f$, we have $\inf_{g \in \Sigma^\omega} \Phi(sg) \leq v$.*

Next, we present some examples of liveness and co-liveness properties. We start by showing that lim inf- and lim sup-properties can be live and co-live.

Example 40. Let $\Sigma = \{a, b\}$ be an alphabet, and let $P = \Box\Diamond a$ and $Q = \Diamond\Box b$ be boolean properties defined in linear temporal logic. Consider their characteristic properties Φ_P and Φ_Q. As we pointed out earlier, our definitions generalize their boolean counterparts, therefore Φ_P and Φ_Q are both live and co-live. Moreover, Φ_P is a lim sup-property: define $\pi_P(s) = 1$ if $s \in \Sigma^* a$, and $\pi_P(s) = 0$ otherwise, and observe that $\Phi_P(f) = \limsup_{s \prec f} \pi_P(s)$ for all $f \in \Sigma^\omega$. Similarly, Φ_Q is a lim inf-property. □

Now, we show that the maximal response-time property is live, and the minimal response time is co-live.

Example 41. Recall the co-safety property $\Phi_{\max}$ of maximal response time from Example 26. Let $f \in \Sigma^\omega$ such that $\Phi_{\max}(f) < \infty$. We can extend every prefix $s \prec f$ with $g = \mathtt{rq}\,\mathtt{tk}^\omega$, which gives us $\Phi_{\max}(sg) = \infty > \Phi(f)$. Equivalently, for every $f \in \Sigma^\omega$, we have $\Phi^*_{\max}(f) = \infty > \Phi_{\max}(f)$. Hence $\Phi_{\max}$ is live and, analogously, the safety property $\Phi_{\min}$ from Example 3 is co-live. □

Finally, we show that the *average response-time* property is live and co-live.

Example 42. Let $\Sigma = \{\mathtt{rq}, \mathtt{gr}, \mathtt{tk}, \mathtt{oo}\}$. For all $s \in \Sigma^*$, let $p(s) = 1$ if there is no pending $\mathtt{rq}$ in s, and $p(s) = 0$ otherwise. Define $\pi_{\text{valid}}(s) = |\{r \preceq s \mid \exists t \in \Sigma^* : r = t\,\mathtt{rq} \wedge p(t) = 1\}|$ as the number of valid requests in s, and define $\pi_{\text{time}}(s)$ as the number of $\mathtt{tk}$ observations that occur after a valid $\mathtt{rq}$ and before the matching $\mathtt{gr}$. Then, $\Phi_{\text{avg}} = (\pi_{\text{avg}}, \liminf)$, where $\pi_{\text{avg}}(s) = \frac{\pi_{\text{time}}(s)}{\pi_{\text{valid}}(s)}$ for all $s \in \Sigma^*$ with $\pi_{\text{valid}}(s) > 0$, and $\pi_{\text{avg}}(s) = \infty$ otherwise. For example, $\pi_{\text{avg}}(s) = \frac{3}{2}$ for $s = \mathtt{rq}\,\mathtt{tk}\,\mathtt{gr}\,\mathtt{tk}\,\mathtt{rq}\,\mathtt{tk}\,\mathtt{rq}\,\mathtt{tk}$. Note that Φ_{avg} is a lim inf-property.

The property Φ_{avg} is defined on the value domain $[0, \infty]$ and is both live and co-live. To see this, let $f \in \Sigma^\omega$ such that $0 < \Phi_{\text{avg}}(f) < \infty$ and, for every prefix $s \prec f$, consider $g = \mathtt{rq}\,\mathtt{tk}^\omega$ and $h = \mathtt{gr}\,(\mathtt{rq}\,\mathtt{gr})^\omega$. Since sg has a pending request followed by infinitely many clock ticks, we have $\Phi_{\text{avg}}(sg) = \infty$. Similarly, since sh eventually has all new requests immediately granted, we get $\Phi_{\text{avg}}(sh) = 0$. □

5.1 The Quantitative Safety-Liveness Decomposition

A celebrated theorem states that every boolean property can be expressed as an intersection of a safety property and a liveness property [4]. In this section, we prove the analogous result for the quantitative setting.

Example 43. Let $\Sigma = \{\texttt{rq}, \texttt{gr}, \texttt{tk}, \texttt{oo}\}$. Recall the maximal response-time property $\Phi_{\max}$ from Example 26, and the average response-time property Φ_{avg} from Example 42. Let $n > 0$ be an integer and define a new property Φ by $\Phi(f) = \Phi_{\text{avg}}(f)$ if $\Phi_{\max}(f) \leq n$, and $\Phi(f) = 0$ otherwise. For the safety closure of Φ, we have $\Phi^*(f) = n$ if $\Phi_{\max}(f) \leq n$, and $\Phi^*(f) = 0$ otherwise. Now, we further define $\Psi(f) = \Phi_{\text{avg}}(f)$ if $\Phi_{\max}(f) \leq n$, and $\Psi(f) = n$ otherwise. Observe that Ψ is live, because every prefix of a trace whose value is less than n can be extended to a greater value. Finally, note that for all $f \in \Sigma^\omega$, we can express $\Phi(f)$ as the pointwise minimum of $\Phi^*(f)$ and $\Psi(f)$. Intuitively, the safety part Φ^* of this decomposition checks whether the maximal response time stays below the permitted bound, and the liveness part Ψ keeps track of the average response time as long as the bound is satisfied. □

Following a similar construction, we show that a safety-liveness decomposition exists for every property.

Theorem 44. *For every property Φ, there exists a liveness property Ψ such that $\Phi(f) = \min(\Phi^*(f), \Psi(f))$ for all $f \in \Sigma^\omega$.*

In particular, if the given property is safe or live, the decomposition is trivial.

Remark 45. Let Φ be a property. If Φ is safe (resp. live), then the safety (resp. liveness) part of the decomposition is Φ itself, and the liveness (resp. safety) part is the constant property that maps every trace to $\top$.

For co-safety and co-liveness, the duals of Theorem 44 and Remark 45 hold. In particular, every property is the pointwise maximum of its co-safety closure and a co-liveness property.

5.2 Related Definitions of Quantitative Liveness

In [41], the authors define a property Φ as *multi-live* iff $\Phi^*(f) > \bot$ for all $f \in \Sigma^\omega$. We show that our definition is more restrictive, resulting in fewer liveness properties while still allowing a safety-liveness decomposition.

Proposition 46. *Every live property is multi-live, and the inclusion is strict.*

We provide a separating example on a totally ordered domain below.

Example 47. Let $\Sigma = \{a, b, c\}$, and consider the following property: $\Phi(f) = 0$ if $f \models \Box a$, and $\Phi(f) = 1$ if $f \models \Diamond c$, and $\Phi(f) = 2$ otherwise (i.e., if $f \models \Diamond b \wedge \Box\neg c$). For all $f \in \Sigma^\omega$ and prefixes $s \prec f$, we have $\Phi(sc^\omega) = 1$. Thus $\Phi^*(f) \neq \bot$, which implies that Φ is multi-live. However, Φ is not live. Indeed, for every $f \in \Sigma^\omega$ such that $f \models \Diamond c$, we have $\Phi(f) = 1 < \top$. Moreover, f admits some prefix s that contains an occurrence of c, thus satisfying $\sup_{g \in \Sigma^\omega} \Phi(sg) = 1$. □

In [27], the authors define a property Φ as *verdict-live* iff for every $f \in \Sigma^\omega$ and value $v \not\leq \Phi(f)$, every prefix $s \prec f$ satisfies $\Phi(sg) = v$ for some $g \in \Sigma^\omega$. We show that our definition is more liberal.

Proposition 48. *Every verdict-live property is live, and the inclusion is strict.*

We provide a separating example below, concluding that our definition is strictly more general even for totally ordered domains.

Example 49. Let $\Sigma = \{a, b\}$, and consider the following property: $\Phi(f) = 0$ if $f \not\models \Diamond b$, and $\Phi(f) = 1$ if $f \models \Diamond(b \wedge \bigcirc\Diamond b)$, and $\Phi(f) = 2^{-|s|}$ otherwise, where $s \prec f$ is the shortest prefix in which b occurs. Consider an arbitrary $f \in \Sigma^\omega$. If $\Phi(f) = 1$, then the liveness condition is vacuously satisfied. If $\Phi(f) = 0$, then $f = a^\omega$, and every prefix $s \prec f$ can be extended with $g = ba^\omega$ or $h = b^\omega$ to obtain $\Phi(sg) = 2^{-(|s|+1)}$ and $\Phi(sh) = 1$. If $0 < \Phi(f) < 1$, then f satisfies $\Diamond b$ but not $\Diamond(b \wedge \bigcirc\Diamond b)$, and every prefix $s \prec f$ can be extended with b^ω to obtain $\Phi(sb^\omega) = 1$. Hence Φ is live. However, Φ is not verdict-live. To see this, consider the trace $f = a^k b a^\omega$ for some integer $k \geq 1$ and note that $\Phi(f) = 2^{-(k+1)}$. Although all prefixes of f can be extended to reach the value 1, the value domain contains elements between $\Phi(f)$ and 1, namely the values 2^{-m} for $1 \leq m \leq k$. Each of these values can be rejected after reading a finite prefix of f, because for $n \geq m$ it is not possible to extend a^n to reach the value 2^{-m}. □

6 Approximate Monitoring through Approximate Safety

In this section, we consider properties on extended reals $\mathbb{R}^{\pm\infty} = \mathbb{R} \cup \{-\infty, +\infty\}$. We denote by $\mathbb{R}_{\geq 0}$ the set of nonnegative real numbers.

Definition 50 (Approximate safety and co-safety). *Let $\alpha \in \mathbb{R}_{\geq 0}$. A property Φ is* α-safe *iff for every $f \in \Sigma^\omega$ and value $v \in \mathbb{R}^{\pm\infty}$ with $\Phi(f) < v$, there exists a prefix $s \prec f$ such that $\sup_{g \in \Sigma^\omega} \Phi(sg) < v + \alpha$. Similarly, Φ is* α-co-safe *iff for every $f \in \Sigma^\omega$ and $v \in \mathbb{R}^{\pm\infty}$ with $\Phi(f) > v$, there exists $s \prec f$ such that $\inf_{g \in \Sigma^\omega} \Phi(sg) > v - \alpha$. When Φ is α-safe (resp. α-co-safe) for some $\alpha \in \mathbb{R}_{\geq 0}$, we say that Φ is* approximately safe *(resp.* approximately co-safe*).*

Approximate safety can be characterized through the following relation with the safety closure.

Proposition 51. *For every error bound $\alpha \in \mathbb{R}_{\geq 0}$, a property Φ is α-safe iff $\Phi^*(f) - \Phi(f) \leq \alpha$ for all $f \in \Sigma^\omega$.*

An analogue of Proposition 51 holds for approximate co-safety and the co-safety closure. Moreover, approximate safety and approximate co-safety are dual notions that are connected by the complement operation, similarly to their precise counterparts (Theorem 27).

6.1 The Intersection of Approximate Safety and Co-safety

Recall the ghost monitor from the introduction. If, after a finite number of observations, all the possible prediction values are close enough, then we can simply freeze the current value and achieve a sufficiently small error. This happens for properties that are both approximately safe and approximately co-safe, generalizing the unfolding approximation of discounted properties [13].

Proposition 52. *For every limit property Φ and all error bounds $\alpha, \beta \in \mathbb{R}_{\geq 0}$, if Φ is α-safe and β-co-safe, then the set $S_\delta = \{s \in \Sigma^* \mid \sup_{r_1 \in \Sigma^*} \Phi(sr_1) - \inf_{r_2 \in \Sigma^*} \Phi(sr_2) \geq \delta\}$ is finite for all reals $\delta > \alpha + \beta$.*

Based on this proposition, we show that, for limit properties that are both approximately safe and approximately co-safe, the influence of the suffix on the property value is eventually negligible.

Theorem 53. *For every limit property Φ such that $\Phi(f) \in \mathbb{R}$ for all $f \in \Sigma^\omega$, and for all error bounds $\alpha, \beta \in \mathbb{R}_{\geq 0}$, if Φ is α-safe and β-co-safe, then for every real $\delta > \alpha + \beta$ and trace $f \in \Sigma^\omega$, there is a prefix $s \prec f$ such that for all continuations $w \in \Sigma^* \cup \Sigma^\omega$, we have $|\Phi(sw) - \Phi(s)| < \delta$.*

We illustrate this theorem with a *discounted safety* property.

Example 54. Let $P \subseteq \Sigma^\omega$ be a boolean safety property. We define the finitary property $\pi_P : \Sigma^* \to [0,1]$ as follows: $\pi_P(s) = 1$ if $sf \in P$ for some $f \in \Sigma^\omega$, and $\pi_P(s) = 1 - 2^{-|r|}$ otherwise, where $r \preceq s$ is the shortest prefix with $rf \notin P$ for all $f \in \Sigma^\omega$. The limit property $\Phi = (\pi_P, \inf)$ is called *discounted safety* [3]. Because Φ is an inf-property, it is safe by Theorem 20. Now consider the finitary property π'_P defined by $\pi'_P(s) = 1 - 2^{-|s|}$ if $sf \in P$ for some $f \in \Sigma^\omega$, and $\pi'_P(s) = 1 - 2^{-|r|}$ otherwise, where $r \preceq s$ is the shortest prefix with $rf \notin P$ for all $f \in \Sigma^\omega$. Let $\Phi' = (\pi'_P, \sup)$, and note that $\Phi(f) = \Phi'(f)$ for all $f \in \Sigma^\omega$. Hence Φ is also co-safe, because it is a sup-property.

Let $f \in \Sigma^\omega$ and $\delta > 0$. For every prefix $s \prec f$, the set of possible prediction values is either the range $[1 - 2^{-|s|}, 1]$ or the singleton $\{1 - 2^{-|r|}\}$, where $r \preceq s$ is chosen as above. In the latter case, we have $|\Phi(sw) - \Phi(s)| = 0 < \delta$ for all $w \in \Sigma^* \cup \Sigma^\omega$. In the former case, since the range becomes smaller as the prefix grows, there is a prefix $s' \prec f$ with $2^{-|s'|} < \delta$, which yields $|\Phi(s'w) - \Phi(s')| < \delta$ for all $w \in \Sigma^* \cup \Sigma^\omega$. □

6.2 Finite-state Approximate Monitoring

Monitors with finite state spaces are particularly desirable, because finite automata enjoy a plethora of desirable closure and decidability properties. Here, we prove that properties that are both approximately safe and approximately co-safe can be monitored approximately by a finite-state monitor. First, we recall the notion of abstract quantitative monitor from [30].

A binary relation $\sim$ over Σ^* is an *equivalence relation* iff it is reflexive, symmetric, and transitive. Such a relation is *right-monotonic* iff $s_1 \sim s_2$ implies $s_1 r \sim s_2 r$ for all $s_1, s_2, r \in \Sigma^*$. For an equivalence relation $\sim$ over Σ^* and a finite trace $s \in \Sigma^*$, we write $[s]_\sim$ for the equivalence class of $\sim$ to which s belongs. When $\sim$ is clear from the context, we write $[s]$ instead. We denote by $\Sigma^*/\!\sim$ the quotient of the relation $\sim$.

Definition 55 (Abstract monitor [30]). *An* abstract monitor $\mathcal{M} = (\sim, \gamma)$ *is a pair consisting of a right-monotonic equivalence relation $\sim$ on Σ^* and a function $\gamma\colon (\Sigma^*/\!\sim) \to \mathbb{R}^{\pm\infty}$. The monitor $\mathcal{M}$ is* finite-state *iff the relation*

$\sim$ has finitely many equivalence classes. Let $\delta_{\text{fin}}, \delta_{\text{lim}} \in \mathbb{R}^{\pm\infty}$ be error bounds. We say that $\mathcal{M}$ is a $(\delta_{\text{fin}}, \delta_{\text{lim}})$-monitor *for a given limit property $\Phi = (\pi, \ell)$ iff for all $s \in \Sigma^*$ and $f \in \Sigma^\omega$, we have $|\pi(s) - \gamma([s])| \leq \delta_{\text{fin}}$ and $|\ell_{s \prec f}(\pi(s)) - \ell_{s \prec f}(\gamma([s]))| \leq \delta_{\text{lim}}$.*

Building on Theorem 53, we identify a sufficient condition to guarantee the existence of an abstract monitor with finitely many equivalence classes.

Theorem 56. *For every limit property Φ such that $\Phi(f) \in \mathbb{R}$ for all $f \in \Sigma^\omega$, and for all error bounds $\alpha, \beta \in \mathbb{R}_{\geq 0}$, if Φ is α-safe and β-co-safe, then for every real $\delta > \alpha + \beta$, there exists a finite-state (δ, δ)-monitor for Φ.*

Due to Theorem 56, the discounted safety property of Example 54 has a finite-state monitor for every positive error bound. We remark that Theorem 56 is proved by a construction that generalizes the unfolding approach for the approximate determinization of discounted automata [12], which unfolds an automaton until the distance constraint is satisfied.

7 Conclusion

We presented a generalization of safety and liveness that lifts the safety-progress hierarchy to the quantitative setting of [18] while preserving major desirable features of the boolean setting, such as the safety-liveness decomposition.

Monitorability identifies a boundary separating properties that can be verified or falsified from a finite number of observations, from those that cannot. Safety-liveness and co-safety-co-liveness decompositions allow us separate, for an individual property, monitorable parts from nonmonitorable parts. The larger the monitorable parts of the given property, the stronger the decomposition. We provided the strongest known safety-liveness decomposition, which consists of a pointwise minimum between a safe part defined by a quantitative safety closure, and a live part which corrects for the difference. We then defined approximate safety as the relaxation of safety by a parametric error bound. This further increases the monitorability of properties and offers monitorability at a parametric cost. In fact, we showed that every property that is both approximately safe and approximately co-safe can be monitored arbitrarily precisely by a finite-state monitor. A future direction is to extend our decomposition to approximate safety together with a support for quantitative assumptions [32].

The literature contains efficient model-checking procedures that leverage the boolean safety hypothesis [36,40]. We thus expect that also quantitative safety and co-safety, and their approximations, enable efficient verification algorithms for quantitative properties.

Acknowledgments. We thank the anonymous reviewers for their helpful comments. This work was supported in part by the ERC-2020-AdG 101020093.

References

1. de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: Model checking discounted temporal properties. Theor. Comput. Sci. **345**(1), 139–170 (2005). https://doi.org/10.1016/j.tcs.2005.07.033
2. de Alfaro, L., Faella, M., Stoelinga, M.: Linear and branching metrics for quantitative transition systems. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3142, pp. 97–109. Springer (2004). https://doi.org/10.1007/978-3-540-27836-8_11
3. de Alfaro, L., Henzinger, T.A., Majumdar, R.: Discounting the future in systems theory. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings. Lecture Notes in Computer Science, vol. 2719, pp. 1022–1037. Springer (2003). https://doi.org/10.1007/3-540-45061-0_79
4. Alpern, B., Schneider, F.B.: Defining liveness. Inf. Process. Lett. **21**(4), 181–185 (1985). https://doi.org/10.1016/0020-0190(85)90056-0
5. Alpern, B., Schneider, F.B.: Recognizing safety and liveness. Distributed Comput. **2**(3), 117–126 (1987). https://doi.org/10.1007/BF01782772
6. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification - Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457, pp. 1–33. Springer (2018). https://doi.org/10.1007/978-3-319-75632-5_1
7. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. J. Log. Comput. **20**(3), 651–674 (2010). https://doi.org/10.1093/logcom/exn075
8. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4), 14:1–14:64 (2011). https://doi.org/10.1145/2000799.2000800
9. Bloem, R., Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Better quality in synthesis through quantitative objectives. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5643, pp. 140–156. Springer (2009). https://doi.org/10.1007/978-3-642-02658-4_14
10. Bloem, R., Chatterjee, K., Jobstmann, B.: Graph games and reactive synthesis. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 921–962. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_27
11. Boker, U., Chatterjee, K., Henzinger, T.A., Kupferman, O.: Temporal specifications with accumulative values. ACM Trans. Comput. Log. **15**(4), 27:1–27:25 (2014). https://doi.org/10.1145/2629686
12. Boker, U., Henzinger, T.A.: Approximate determinization of quantitative automata. In: D'Souza, D., Kavitha, T., Radhakrishnan, J. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India. LIPIcs, vol. 18, pp. 362–373. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2012). https://doi.org/10.4230/LIPIcs.FSTTCS.2012.362
13. Boker, U., Henzinger, T.A.: Exact and approximate determinization of discounted-sum automata. Log. Methods Comput. Sci. **10**(1) (2014). https://doi.org/10.2168/LMCS-10(1:10)2014

14. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N.: Quantitative analysis of real-time systems using priced timed automata. Commun. ACM **54**(9), 78–87 (2011). https://doi.org/10.1145/1995376.1995396
15. Bouyer, P., Markey, N., Randour, M., Larsen, K.G., Laursen, S.: Average-energy games. Acta Informatica **55**(2), 91–127 (2018). https://doi.org/10.1007/s00236-016-0274-1
16. Cerný, P., Henzinger, T.A., Radhakrishna, A.: Simulation distances. Theor. Comput. Sci. **413**(1), 21–35 (2012). https://doi.org/10.1016/j.tcs.2011.08.002
17. Chang, E., Manna, Z., Pnueli, A.: The safety-progress classification. In: Bauer, F.L., Brauer, W., Schwichtenberg, H. (eds.) Logic and Algebra of Specification. pp. 143–202. Springer Berlin Heidelberg, Berlin, Heidelberg (1993). https://doi.org/10.1007/978-3-642-58041-3_5
18. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Trans. Comput. Log. **11**(4), 23:1–23:38 (2010). https://doi.org/10.1145/1805950.1805953
19. Chatterjee, K., Henzinger, T.A., Otop, J.: Nested weighted automata. ACM Trans. Comput. Log. **18**(4), 31:1–31:44 (2017). https://doi.org/10.1145/3152769
20. D'Antoni, L., Samanta, R., Singh, R.: Qlose: Program repair with quantitative objectives. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9780, pp. 383–401. Springer (2016). https://doi.org/10.1007/978-3-319-41540-6_21
21. Fahrenberg, U., Legay, A.: Generalized quantitative analysis of metric transition systems. In: Shan, C. (ed.) Programming Languages and Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8301, pp. 192–208. Springer (2013). https://doi.org/10.1007/978-3-319-03542-0_14
22. Fahrenberg, U., Legay, A.: The quantitative linear-time-branching-time spectrum. Theor. Comput. Sci. **538**, 54–69 (2014). https://doi.org/10.1016/j.tcs.2013.07.030
23. Falcone, Y., Fernandez, J., Mounier, L.: What can you verify and enforce at runtime? Int. J. Softw. Tools Technol. Transf. **14**(3), 349–382 (2012). https://doi.org/10.1007/s10009-011-0196-8
24. Faran, R., Kupferman, O.: Spanning the spectrum from safety to liveness. Acta Informatica **55**(8), 703–732 (2018). https://doi.org/10.1007/s00236-017-0307-4
25. Ferrère, T., Henzinger, T.A., Kragl, B.: Monitoring event frequencies. In: Fernández, M., Muscholl, A. (eds.) 28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain. LIPIcs, vol. 152, pp. 20:1–20:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.CSL.2020.20
26. Ferrère, T., Henzinger, T.A., Saraç, N.E.: A theory of register monitors. In: Dawar, A., Grädel, E. (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018. pp. 394–403. ACM (2018). https://doi.org/10.1145/3209108.3209194
27. Gorostiaga, F., Sánchez, C.: Monitorability of expressive verdicts. In: Deshmukh, J.V., Havelund, K., Perez, I. (eds.) NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13260, pp. 693–712. Springer (2022). https://doi.org/10.1007/978-3-031-06773-0_37
28. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: Katoen, J., Stevens, P. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of

the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2280, pp. 342–356. Springer (2002). https://doi.org/10.1007/3-540-46002-0_24
29. Henzinger, T.A.: Quantitative reactive modeling and verification. Comput. Sci. Res. Dev. **28**(4), 331–344 (2013). https://doi.org/10.1007/s00450-013-0251-7
30. Henzinger, T.A., Mazzocchi, N., Saraç, N.E.: Abstract monitors for quantitative specifications. In: Dang, T., Stolz, V. (eds.) Runtime Verification - 22nd International Conference, RV 2022, Tbilisi, Georgia, September 28-30, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13498, pp. 200–220. Springer (2022). https://doi.org/10.1007/978-3-031-17196-3_11
31. Henzinger, T.A., Otop, J.: From model checking to model measuring. In: D'Argenio, P.R., Melgratti, H.C. (eds.) CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8052, pp. 273–287. Springer (2013). https://doi.org/10.1007/978-3-642-40184-8_20
32. Henzinger, T.A., Saraç, N.E.: Monitorability under assumptions. In: Deshmukh, J., Nickovic, D. (eds.) Runtime Verification - 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6-9, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12399, pp. 3–18. Springer (2020). https://doi.org/10.1007/978-3-030-60508-7_1
33. Henzinger, T.A., Saraç, N.E.: Quantitative and approximate monitoring. In: 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021. pp. 1–14. IEEE (2021). https://doi.org/10.1109/LICS52264.2021.9470547
34. Katoen, J., Song, L., Zhang, L.: Probably safe or live. In: Henzinger, T.A., Miller, D. (eds.) Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014. pp. 55:1–55:10. ACM (2014). https://doi.org/10.1145/2603088.2603147
35. Kim, M., Kannan, S., Lee, I., Sokolsky, O., Viswanathan, M.: Computational analysis of run-time monitoring - fundamentals of java-mac. In: Havelund, K., Rosu, G. (eds.) Runtime Verification 2002, RV 2002, FLoC Satellite Event, Copenhagen, Denmark, July 26, 2002. Electronic Notes in Theoretical Computer Science, vol. 70, pp. 80–94. Elsevier (2002). https://doi.org/10.1016/S1571-0661(04)80578-4
36. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods Syst. Des. **19**(3), 291–314 (2001). https://doi.org/10.1023/A:1011254632723
37. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic Model Checking: Advances and Applications, pp. 73–121. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-57685-5_3
38. Kwiatkowska, M.Z.: Quantitative verification: models techniques and tools. In: Crnkovic, I., Bertolino, A. (eds.) Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007. pp. 449–458. ACM (2007). https://doi.org/10.1145/1287624.1287688
39. Lamport, L.: Proving the correctness of multiprocess programs. IEEE Trans. Software Eng. **3**(2), 125–143 (1977). https://doi.org/10.1109/TSE.1977.229904
40. Latvala, T.: Efficient model checking of safety properties. In: Ball, T., Rajamani, S.K. (eds.) Model Checking Software, 10th International SPIN Workshop. Portland, OR, USA, May 9-10, 2003, Proceedings. Lecture Notes in Computer Science,

vol. 2648, pp. 74–88. Springer (2003). https://doi.org/10.1007/3-540-44829-2_5
41. Li, Y., Droste, M., Lei, L.: Model checking of linear-time properties in multi-valued systems. Inf. Sci. **377**, 51–74 (2017). https://doi.org/10.1016/j.ins.2016.10.030
42. Manna, Z., Pnueli, A.: Adequate proof principles for invariance and liveness properties of concurrent programs. Sci. Comput. Program. **4**(3), 257–289 (1984). https://doi.org/10.1016/0167-6423(84)90003-0
43. Peled, D., Havelund, K.: Refining the safety-liveness classification of temporal properties according to monitorability. In: Margaria, T., Graf, S., Larsen, K.G. (eds.) Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 11200, pp. 218–234. Springer (2018). https://doi.org/10.1007/978-3-030-22348-9_14
44. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4085, pp. 573–586. Springer (2006). https://doi.org/10.1007/11813040_38
45. Qian, J., Shi, F., Cai, Y., Pan, H.: Approximate safety properties in metric transition systems. IEEE Trans. Reliab. **71**(1), 221–234 (2022). https://doi.org/10.1109/TR.2021.3139616
46. Sistla, A.P.: Safety, liveness and fairness in temporal logic. Formal Aspects Comput. **6**(5), 495–512 (1994). https://doi.org/10.1007/BF01211865
47. Thrane, C.R., Fahrenberg, U., Larsen, K.G.: Quantitative analysis of weighted transition systems. J. Log. Algebraic Methods Program. **79**(7), 689–703 (2010). https://doi.org/10.1016/j.jlap.2010.07.010
48. Weiner, S., Hasson, M., Kupferman, O., Pery, E., Shevach, Z.: Weighted safety. In: Hung, D.V., Ogawa, M. (eds.) Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8172, pp. 133–147. Springer (2013). https://doi.org/10.1007/978-3-319-02444-8_11

On the Comparison of Discounted-Sum Automata with Multiple Discount Factors

Udi Boker* and Guy Hefetz(✉)

Reichman University, Herzliya, Israel
udiboker@runi.ac.il, ghefetz@gmail.com

Abstract. We look into the problems of comparing nondeterministic discounted-sum automata on finite and infinite words. That is, the problems of checking for automata $\mathcal{A}$ and $\mathcal{B}$ whether or not it holds that for all words w, $\mathcal{A}(w) = \mathcal{B}(w)$, $\mathcal{A}(w) \leq \mathcal{B}(w)$, or $\mathcal{A}(w) < \mathcal{B}(w)$.
These problems are known to be decidable when both automata have the same single integral discount factor, while decidability is open in all other settings: when the single discount factor is a non-integral rational; when each automaton can have multiple discount factors; and even when each has a single integral discount factor, but the two are different.
We show that it is undecidable to compare discounted-sum automata with multiple discount factors, even if all are integrals, while it is decidable to compare them if each has a single, possibly different, integral discount factor. To this end, we also provide algorithms to check for given nondeterministic automaton $\mathcal{N}$ and deterministic automaton $\mathcal{D}$, each with a single, possibly different, rational discount factor, whether or not $\mathcal{N}(w) = \mathcal{D}(w)$, $\mathcal{N}(w) \geq \mathcal{D}(w)$, or $\mathcal{N}(w) > \mathcal{D}(w)$ for all words w.

Keywords: Discounted-sum Automata · Comparison · Containment

1 Introduction

Equivalence and containment checks of Boolean automata, namely the checks of whether $L(\mathcal{A}) = L(\mathcal{B})$, $L(\mathcal{A}) \subseteq L(\mathcal{B})$, or $L(\mathcal{A}) \subset L(\mathcal{B})$, where $L(\mathcal{A})$ and $L(\mathcal{B})$ are the languages that $\mathcal{A}$ and $\mathcal{B}$ recognize, are central in the usage of automata theory in diverse areas, and in particular in formal verification (e.g, [34,26,17,33,35,28]). Likewise, comparison of quantitative automata, which extends the equivalence and containment checks by asking whether $\mathcal{A}(w) = \mathcal{B}(w)$, whether $\mathcal{A}(w) \leq \mathcal{B}(w)$, or whether $\mathcal{A}(w) < \mathcal{B}(w)$ for all words w, are essential for harnessing quantitative-automata theory to the service of diverse fields and in particular to the service of quantitative formal verification (e.g, [15,14,21,11,27,3,5,22]).

Discounted summation is a common valuation function in quantitative automata theory (e.g, [19,12,14,15]), as well as in various other computational models, such as games (e.g., [37,4,1]), Markov decision processes (e.g, [23,29,16]), and reinforcement learning (e.g, [32,36]), as it formalizes the concept that an immediate reward is better than a potential one in the far future, as well as that a

* Research supported by the Israel Science Foundation grant 2410/22.

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 371–391, 2023.
https://doi.org/10.1007/978-3-031-30829-1_18

potential problem (such as a bug in a reactive system) in the far future is less troubling than a current one.

A nondeterministic discounted-sum automaton (NDA) has rational weights on the transitions, and a fixed rational discount factor $\lambda > 1$. The value of a (finite or infinite) run is the discounted summation of the weights on the transitions, such that the weight in the ith transition of the run is divided by λ^i. The value of a (finite or infinite) word is the infimum value of the automaton runs on it. An NDA thus realizes a function from words to real numbers.

NDAs cannot always be determinized [15], they are not closed under basic algebraic operations [8], and their comparison is not known to be decidable, relating to various longstanding open problems [9]. However, restricting NDAs to have an integral discount factor $\lambda \in \mathbb{N} \setminus \{0, 1\}$ provides a robust class of automata that is closed under determinization and under algebraic operations, and for which comparison is decidable [8].

Various variants of NDAs are studied in the literature, among which are *functional*, *k-valued*, *probabilistic*, and more [21,20,13]. Yet, until recently, all of these models were restricted to have a single discount factor. This is a significant restriction of the general discounted-summation paradigm, in which multiple discount factors are considered. For example, Markov decision processes and discounted-sum games allow multiple discount factors within the same entity [23,4]. In [6], NDAs were extended to NMDAs, allowing for multiple discount factors, where each transition can have a different one. Special attention was given to integral NMDAs, namely to those with only integral discount factors, analyzing whether they preserve the good properties of integral NDAs. It was shown that they are generally not closed under determinization and under algebraic operations, while a restricted class of them, named tidy-NMDAs, in which the choice of discount factors depends on the prefix of the word read so far, does preserve the good properties of integral NDAs.

While comparison of tidy-NMDAs with the same choice function is decidable in PSPACE [6], it was left open whether comparison of general integral NMDAs $\mathcal{A}$ and $\mathcal{B}$ is decidable. It is even open whether comparison of two integral NDAs with different (single) discount factors is decidable.

We show that it is undecidable to resolve for given NMDA $\mathcal{N}$ and deterministic NMDA (DMDA) $\mathcal{D}$, even if both have only integral discount factors, on both finite and infinite words, whether $\mathcal{N} \equiv \mathcal{D}$ and whether $\mathcal{N} \leq \mathcal{D}$, and on finite words also whether $\mathcal{N} < \mathcal{D}$. We prove the undecidability result by reduction from the halting problem of two-counter machines. The general scheme follows similar reductions, such as in [18,2], yet the crux is in simulating a counter by integral NMDAs. Upfront, discounted summation is not suitable for simulating counters, since a current increment has, in the discounted setting, a much higher influence than of a far-away decrement. However, we show that multiple discount factors allow in a sense to eliminate the influence of time, having automata in which no matter where a letter appears in the word, it will have the same influence on the automaton value. (See Lemma 1 and Fig. 3). Another main part of the proof is in showing how to nondeterministically adjust the automaton weights

and discount factors in order to "detect" whether a counter is at a current value 0. (See Figs. 5, 6, 8 and 9.)

On the positive side, we provide algorithms to decide for given NDA $\mathcal{N}$ and deterministic NDA (DDA) $\mathcal{D}$, with arbitrary, possibly different, rational discount factors, whether $\mathcal{N} \equiv \mathcal{D}$, $\mathcal{N} \geq \mathcal{D}$, or $\mathcal{N} > \mathcal{D}$ (Theorem 4). Our algorithms work on both finite and infinite words, and run in PSPACE when the automata weights are represented in binary and their discount factors in unary. Since integral NDAs can always be determinized [8], our method also provides an algorithm to compare two integral NDAs, though not necessarily in PSPACE, since determinization might exponentially increase the number of states. (Even though determinization of NDAs is in PSPACE [8,6], the exponential number of states might require an exponential space in our algorithms of comparing NDAs with different discount factors.)

The challenge with comparing automata with different discount factors comes from the combination of their different accumulations, which tends to be intractable, resulting in the undecidability of comparing integral NMDAs, and in the open problems of comparing rational NDAs and of analyzing the representation of numbers in a non-integral basis [30,24,25,9]. Yet, the main observation underlying our algorithm is that when each automaton has a single discount factor, we may unfold the combination of their computation trees only up to some level k, after which we can analyze their continuation separately, first handling the automaton with the lower (slower decreasing) discount factor and then the other one. The idea is that after level k, since the accumulated discounting of the second automaton is already much more significant, even a single non-optimal transition of the first automaton cannot be compensated by a continuation that is better with respect to the second automaton. We thus compute the optimal suffix words and runs of the first automaton from level k, on top which we compute the optimal runs of the second automaton.

2 Preliminaries

Words. An *alphabet* Σ is an arbitrary finite set, and a *word* over Σ is a finite or infinite sequence of letters in Σ, with ε for the empty word. We denote the concatenation of a finite word u and a finite or infinite word w by $u \cdot w$, or simply by uw. We define Σ^+ to be the set of all finite words except the empty word, i.e., $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For a word $w = \sigma_0\sigma_1\sigma_2 \cdots$ and indexes $i \leq j$, we denote the *letter at index* i as $w[i] = \sigma_i$, and the *sub-word from* i *to* j as $w[i..j] = \sigma_i\sigma_{i+1} \cdots \sigma_j$.

For a finite word w and letter $\sigma \in \Sigma$, we denote the number of occurrences of σ in w by $\#(\sigma, w)$, and for a set $S \subseteq \Sigma$, we denote $\sum_{\sigma \in S} \#(\sigma, w)$ by $\#(S, w)$.

For a finite or infinite word w and a letter $\sigma \in \Sigma$, we define the *prefix of* w *up to* σ, $\text{PREF}_\sigma(w)$, as the minimal prefix of w that contains a σ letter if there is a σ letter in w or w itself if it does not contain any σ letters. Formally,

$$\text{PREF}_\sigma(w) = \begin{cases} w\big[0..\min\{i \mid w[i] = \sigma\}\big] & \exists i \mid w[i] = \sigma \\ w & \text{otherwise} \end{cases}$$

Automata. A nondeterministic discounted-sum automaton (NDA) [15] is an automaton with rational weights on the transitions, and a fixed rational discount factor $\lambda > 1$. A nondeterministic discounted-sum automaton with multiple discount factors (NMDA) [6] is similar to an NDA, but with possibly a different discount factor on each of its transitions. They are formally defined as follows:

Definition 1 ([6]). *A nondeterministic discounted-sum automaton with multiple discount factors (NMDA), on finite or infinite words, is a tuple* $\mathcal{A} = \langle \Sigma, Q, \iota, \delta, \gamma, \rho \rangle$ *over an alphabet* Σ*, with a finite set of states* Q*, an initial set of states* $\iota \subseteq Q$*, a transition function* $\delta \subseteq Q \times \Sigma \times Q$*, a weight function* $\gamma : \delta \to \mathbb{Q}$*, and a discount-factor function* $\rho : \delta \to \mathbb{Q} \cap (1, \infty)$*, assigning to each transition its discount factor, which is a rational greater than one.* [1]

- *A* run *of* $\mathcal{A}$ *is a sequence of states and alphabet letters,* $p_0, \sigma_0, p_1, \sigma_1, p_2, \cdots$*, such that* $p_0 \in \iota$ *is an initial state, and for every* i*,* $(p_i, \sigma_i, p_{i+1}) \in \delta$*.*
- *The* length *of a run* r*, denoted by* $|r|$*, is* n *for a finite run* $r = p_0, \sigma_0, p_1, \cdots, \sigma_{n-1}, p_n$*, and* ∞ *for an infinite run.*
- *For an index* $i < |r|$*, we define the* i*-th transition of* r *as* $r[i] = (p_i, \sigma_i, p_{i+1})$*, and the prefix run with* i *transitions as* $r[0..i] = p_0, \sigma_0, p_1, \cdots, \sigma_i, p_{i+1}$*.*
- *The* value *of a finite/infinite run* r *is* $\mathcal{A}(r) = \sum_{i=0}^{|r|-1} \left(\gamma\big(r[i]\big) \cdot \prod_{j=0}^{i-1} \frac{1}{\rho\big(r[j]\big)} \right)$*. For example, the value of the run* $r_1 = q_0, a, q_0, a, q_1, b, q_2$ *of* $\mathcal{A}$ *from Fig. 1 is* $\mathcal{A}(r_1) = 1 + \frac{1}{2} \cdot \frac{1}{3} + 2 \cdot \frac{1}{2 \cdot 3} = \frac{3}{2}$*.*
- *The* value *of* $\mathcal{A}$ *on a finite or infinite word* w *is* $\mathcal{A}(w) = \inf\{\mathcal{A}(r) \mid r \text{ is a run of } \mathcal{A} \text{ on } w\}$*.*
- *For every finite run* $r = p_0, \sigma_0, p_1, \cdots, \sigma_{n-1}, p_n$*, we define the* target state *as* $\delta(r) = p_n$ *and the* accumulated discount factor *as* $\rho(r) = \prod_{i=0}^{n-1} \rho\big(r[i]\big)$*.*
- *When all discount factors are integers, we say that* $\mathcal{A}$ *is an* integral *NMDA.*
- *In the case where* $|\iota| = 1$ *and for every* $q \in Q$ *and* $\sigma \in \Sigma$*, we have* $|\{q' \mid (q, \sigma, q') \in \delta\}| \leq 1$*, we say that* $\mathcal{A}$ *is* deterministic*, denoted by DMDA, and view* δ *as a function from words to states.*
- *When the discount factor function* ρ *is constant,* $\rho \equiv \lambda \in \mathbb{Q} \cap (1, \infty)$*, we say that* $\mathcal{A}$ *is a* nondeterministic discounted-sum automaton *(NDA) [15] with discount factor* λ *(a* λ*-NDA). If* $\mathcal{A}$ *is deterministic, it is a* λ*-DDA.*
- *For a state* $q \in Q$*, we write* $\mathcal{A}^q$ *for the NMDA* $\mathcal{A}^q = \langle \Sigma, Q, \{q\}, \delta, \gamma, \rho \rangle$*.*

Counter machines. A two-counter machine [31] $\mathcal{M}$ is a sequence $(l_1, \ldots, l_n)$ of commands, for some $n \in \mathbb{N}$, involving two counters x and y. We refer to $\{1, \ldots, n\}$ as the *locations* of the machine. For every $i \in \{1, \ldots, n\}$ we refer to l_i as the *command in location* i. There are five possible forms of commands:

$$\textsc{inc}(c),\ \textsc{dec}(c),\ \textsc{goto}\ l_k,\ \textsc{if}\ c{=}0\ \textsc{goto}\ l_k\ \textsc{else goto}\ l_{k'},\ \textsc{halt},$$

where $c \in \{x, y\}$ is a counter and $1 \leq k, k' \leq n$ are locations. For not decreasing a zero-valued counter $c \in \{x, y\}$, every $\textsc{dec}(c)$ command is preceded by the

[1] Discount factors are sometimes defined as numbers between 0 and 1, under which setting weights are multiplied by these factors rather than divided by them.

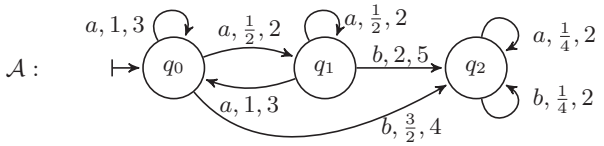


Fig. 1. An NMDA $\mathcal{A}$. The labeling on the transitions indicate the alphabet letter, the weight of the transition, and its discount factor.

command IF c=0 GOTO <CURRENT_LINE> ELSE GOTO <NEXT_LINE>, and there are no other direct goto-commands to it. The counters are initially set to 0. An example of a two-counter machine is given in Fig. 2.

l_1. INC(x)
l_2. INC(x)
l_3. IF x=0 GOTO l_3 ELSE GOTO l_4
l_4. DEC(x)
l_5. IF x=0 GOTO l_6 ELSE GOTO l_3
l_6. HALT

Fig. 2. An example of a two-counter machine.

Let L be the set of possible commands in $\mathcal{M}$, then a *run* of $\mathcal{M}$ is a sequence $\psi = \psi_1, \ldots, \psi_m \in (L \times \mathbb{N} \times \mathbb{N})^*$ such that the following hold:

1. $\psi_1 = \langle l_1, 0, 0 \rangle$.
2. For all $1 < i \leq m$, let $\psi_{i-1} = (l_j, \alpha_x, \alpha_y)$ and $\psi_i = (l', \alpha'_x, \alpha'_y)$. Then, the following hold.
   - If l_j is an INC(x) command (resp. INC(y)), then $\alpha'_x = \alpha_x + 1$, $\alpha'_y = \alpha_y$ (resp. $\alpha_y = \alpha_y + 1$, $\alpha'_x = \alpha_x$), and $l' = l_{j+1}$.
   - If l_j is DEC(x) (resp. DEC(y)) then $\alpha'_x = \alpha_x - 1$, $\alpha'_y = \alpha_y$ (resp. $\alpha_y = \alpha_y - 1$, $\alpha'_x = \alpha_x$), and $l' = l_{j+1}$.
   - If l_j is GOTO l_k then $\alpha'_x = \alpha_x$, $\alpha'_y = \alpha_y$, and $l' = l_k$.
   - If l_j is IF x=0 GOTO l_k ELSE GOTO $l_{k'}$ then $\alpha'_x = \alpha_x$, $\alpha'_y = \alpha_y$, and $l' = l_k$ if $\alpha_x = 0$, and $l' = l_{k'}$ otherwise.
   - If l_j is IF y=0 GOTO l_k ELSE GOTO $l_{k'}$ then $\alpha'_x = \alpha_x$, $\alpha'_y = \alpha_y$, and $l' = l_k$ if $\alpha_y = 0$, and $l' = l_{k'}$ otherwise.
   - If l' is HALT then $i = m$, namely a run does not continue after HALT.

If, in addition, we have that $\psi_m = \langle l_j, \alpha_x, \alpha_y \rangle$ such that l_j is a HALT command, we say that ψ is a *halting run*. We say that a machine $\mathcal{M}$ 0-halts if its run is halting and ends in $\langle l, 0, 0 \rangle$. We say that a sequence of commands $\tau \in L^*$ *fits* a run ψ, if τ is the projection of ψ on its first component.

The *command trace* $\pi = \sigma_1, \ldots, \sigma_m$ of a halting run $\psi = \psi_1, \ldots, \psi_m$ describes the flow of the run, including a description of whether a counter c was equal to 0 or larger than 0 in each occurrence of an IF c=0 GOTO l_k ELSE GOTO $l_{k'}$ command. It is formally defined as follows. σ_m = HALT and for every $1 < i \leq m$, we define σ_{i-1} according to $\psi_{i-1} = (l_j, \alpha_x, \alpha_y)$ in the following manner:

- $\sigma_{i-1} = l_j$ if l_j is not of the form IF c=0 GOTO l_k ELSE GOTO $l_{k'}$.
- $\sigma_{i-1} = (\text{GOTO } l_k, c = 0)$ for $c \in \{x, y\}$, if $\alpha_c = 0$ and the command l_j is of the form IF c=0 GOTO l_k ELSE GOTO $l_{k'}$.
- $\sigma_{i-1} = (\text{GOTO } l_{k'}, c > 0)$ for $c \in \{x, y\}$, if $\alpha_c > 0$ and the command l_j is of the form IF c=0 GOTO l_k ELSE GOTO $l_{k'}$.

For example, the command trace of the halting run of the machine in Fig. 2 is INC(x), INC(x), (GOTO $l_4, x > 0$), DEC(x), (GOTO $l_3, x > 0$), (GOTO $l_4, x > 0$), DEC(x), (GOTO $l_6, x = 0$), HALT.

Deciding whether a given counter machine $\mathcal{M}$ halts is known to be undecidable [31]. Deciding whether $\mathcal{M}$ halts with both counters having value 0, termed the 0-*halting problem*, is also undecidable. Indeed, the halting problem can be reduced to the latter by adding some commands that clear the counters, before every HALT command.

3 Comparison of NMDAs

We show that comparison of (integral) NMDAs is undecidable by reduction from the halting problem of two-counter machines. Notice that our NMDAs only use integral discount factors, while they do have non-integral weights. Yet, weights can be easily changed to integers as well, by multiplying them all by a common denominator and making the corresponding adjustments in the calculations.

We start with a lemma on the accumulated value of certain series of discount factors and weights. Observe that by the lemma, no matter where the pair of discount-factor $\lambda \in \mathbb{N} \setminus \{0, 1\}$ and weight $w = \frac{\lambda-1}{\lambda}$ appear along the run, they will have the same effect on the accumulated value. This property will play a key role in simulating counting by NMDAs.

Lemma 1. *For every sequence $\lambda_1, \cdots, \lambda_m$ of integers larger than 1 and weights $w_1, \cdots, w_m$ such that $w_i = \frac{\lambda_i - 1}{\lambda_i}$, we have $\sum_{i=1}^{m} \left(w_i \cdot \prod_{j=1}^{i-1} \frac{1}{\lambda_j} \right) = 1 - \frac{1}{\prod_{j=1}^{m} \lambda_j}$.*

The proof is by induction on m and appears in [7].

3.1 The Reduction

We turn to our reduction from the halting problem of two-counter machines to the problem of NMDA containment. We provide the construction and the correctness lemma with respect to automata on finite words, and then show in Section 3.2 how to use the same construction also for automata on infinite words.

Given a two-counter machine $\mathcal{M}$ with the commands $(l_1, \ldots, l_n)$, we construct an integral DMDA $\mathcal{A}$ and an integral NMDA $\mathcal{B}$ on finite words, such that $\mathcal{M}$ 0-halts iff there exists a word $w \in \Sigma^+$ such that $\mathcal{B}(w) \geq \mathcal{A}(w)$ iff there exists a word $w \in \Sigma^+$ such that $\mathcal{B}(w) > \mathcal{A}(w)$.

The automata $\mathcal{A}$ and $\mathcal{B}$ operate over the following alphabet Σ, which consists of $5n+5$ letters, standing for the possible elements in a command trace of $\mathcal{M}$:

$$\begin{aligned}
\Sigma^{\text{INCDEC}} &= \{\, \text{INC}(x), \text{DEC}(x), \text{INC}(y), \text{DEC}(y) \,\} \\
\Sigma^{\text{GOTO}} &= \big\{\text{GOTO}\ \ l_k : k \in \{1,\ldots,n\}\big\} \cup \\
&\quad \big\{(\text{GOTO}\ \ l_k, c = 0) : k \in \{1,\ldots,n\}, c \in \{x,y\}\big\} \cup \\
&\quad \big\{(\text{GOTO}\ \ l_{k'}, c > 0) : k' \in \{1,\ldots,n\}, c \in \{x,y\}\big\} \\
\Sigma^{\text{NOHALT}} &= \Sigma^{\text{INCDEC}} \cup \Sigma^{\text{GOTO}} \\
\Sigma &= \Sigma^{\text{NOHALT}} \cup \big\{\text{HALT}\big\}
\end{aligned}$$

When $\mathcal{A}$ and $\mathcal{B}$ read a word $w \in \Sigma^+$, they intuitively simulate a sequence of commands τ_u that induces the command trace $u = \text{PREF}_{\text{HALT}}(w)$. If τ_u fits the actual run of $\mathcal{M}$, and this run 0-halts, then the minimal run of $\mathcal{B}$ on w has a value strictly larger than $\mathcal{A}(w)$. If, however, τ_u does not fit the actual run of $\mathcal{M}$, or it does fit the actual run but it does not 0-halt, then the violation is detected by $\mathcal{B}$, which has a run on w with value strictly smaller than $\mathcal{A}(w)$.

In the construction, we use the following partial discount-factor functions $\rho_p, \rho_d : \Sigma^{\text{NOHALT}} \to \mathbb{N}$ and partial weight functions $\gamma_p, \gamma_d : \Sigma^{\text{NOHALT}} \to \mathbb{Q}$.

$$\rho_p(\sigma) = \begin{cases} 5 & \sigma = \text{INC}(x) \\ 4 & \sigma = \text{DEC}(x) \\ 7 & \sigma = \text{INC}(y) \\ 6 & \sigma = \text{DEC}(y) \\ 15 & \text{otherwise} \end{cases} \qquad \rho_d(\sigma) = \begin{cases} 4 & \sigma = \text{INC}(x) \\ 5 & \sigma = \text{DEC}(x) \\ 6 & \sigma = \text{INC}(y) \\ 7 & \sigma = \text{DEC}(y) \\ 15 & \text{otherwise} \end{cases}$$

$\gamma_p(\sigma) = \frac{\rho_p(\sigma)-1}{\rho_p(\sigma)}$, and $\gamma_d(\sigma) = \frac{\rho_d(\sigma)-1}{\rho_d(\sigma)}$. We say that ρ_p and γ_p are the *primal* discount-factor and weight functions, while ρ_d and γ_d are the *dual* functions. Observe that for every $c \in \{x,y\}$ we have that

$$\rho_p(\text{INC}(c)) = \rho_d(\text{DEC}(c)) > \rho_p(\text{DEC}(c)) = \rho_d(\text{INC}(c)) \tag{1}$$

Intuitively, we will use the primal functions for $\mathcal{A}$'s discount factors and weights, and the dual functions for identifying violations. Notice that if changing the primal functions to the dual ones in more occurrences of $\text{INC}(c)$ letters than of $\text{DEC}(c)$ letters along some run, then by Lemma 1 the run will get a value lower than the original one.

We continue with their formal definitions. $\mathcal{A} = \langle \Sigma, \{q_\mathcal{A}, q^h_\mathcal{A}\}, \{q_\mathcal{A}\}, \delta_\mathcal{A}, \gamma_\mathcal{A}, \rho_\mathcal{A} \rangle$ is an integral DMDA consisting of two states, as depicted in Fig. 3. Observe that the initial state $q_\mathcal{A}$ has self loops for every alphabet letter in Σ^{NOHALT} with weights and discount factors according to the primal functions, and a transition $(q_\mathcal{A}, \text{HALT}, q^h_\mathcal{A})$ with weight of $\frac{14}{15}$ and a discount factor of 15.

The integral NMDA $\mathcal{B} = \langle \Sigma, Q_\mathcal{B}, \iota_\mathcal{B}, \delta_\mathcal{B}, \gamma_\mathcal{B}, \rho_\mathcal{B} \rangle$ is the union of the following eight gadgets (checkers), each responsible for checking a certain type of violation in the description of a 0-halting run of $\mathcal{M}$. It also has the states $q_{\mathsf{freeze}}, q_{\mathsf{halt}} \in Q_\mathcal{B}$

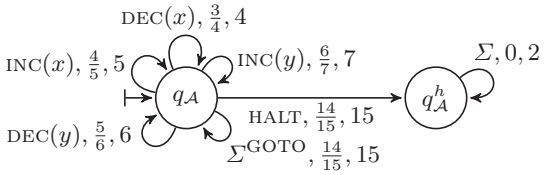


Fig. 3. The DMDA $\mathcal{A}$ constructed for the proof of Lemma 2.

such that for all $\sigma \in \Sigma$, there are 0-weighted transitions $(q_{\mathsf{freeze}}, \sigma, q_{\mathsf{freeze}}) \in \delta_{\mathcal{B}}$ and $(q_{\mathsf{halt}}, \sigma, q_{\mathsf{halt}}) \in \delta_{\mathcal{B}}$ with an arbitrary discount factor. Observer that in all of $\mathcal{B}$'s gadgets, the transition over the letter HALT to q_{halt} has a weight higher than the weight of the corresponding transition in $\mathcal{A}$, so that when no violation is detected, the value of $\mathcal{B}$ on a word is higher than the value of $\mathcal{A}$ on it.

1. Halt Checker. This gadget, depicted in Fig. 4, checks for violations of non-halting runs. Observe that its initial state q_{HC} has self loops identical to those of $\mathcal{A}$'s initial state, a transition to q_{halt} over HALT with a weight higher than the corresponding weight in $\mathcal{A}$, and a transition to the state q_{last} over every letter that is not HALT, "guessing" that the run ends without a HALT command.

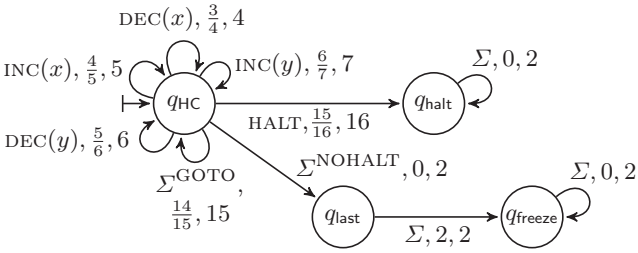


Fig. 4. The Halt Checker in the NMDA $\mathcal{B}$.

2. Negative-Counters Checker. The second gadget, depicted in Fig. 5, checks that the input prefix u has no more DEC(c) than INC(c) commands for each counter $c \in \{x, y\}$. It is similar to $\mathcal{A}$, however having self loops in its initial states that favor DEC(c) commands when compared to $\mathcal{A}$.

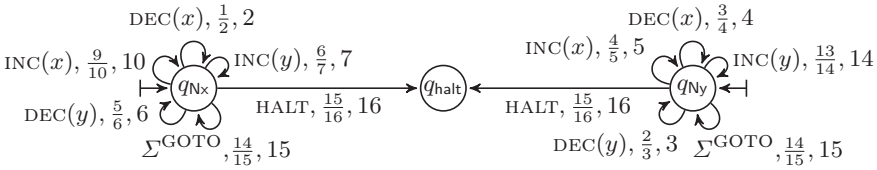


Fig. 5. The negative-counters checker, on the left for x and on the right for y, in the NMDA $\mathcal{B}$.

3. Positive-Counters Checker. The third gadget, depicted in Fig. 6, checks that for every $c \in \{x, y\}$, the input prefix u has no more $\text{INC}(c)$ than $\text{DEC}(c)$ commands. It is similar to $\mathcal{A}$, while having self loops in its initial state according to the dual functions rather than the primal ones.

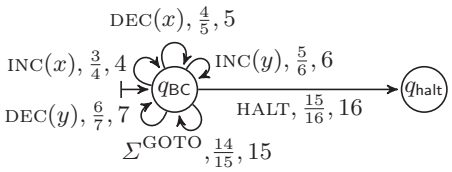


Fig. 6. The Positive-Counters Checker in the NMDA $\mathcal{B}$.

4. Command Checker. The next gadget checks for local violations of successive commands. That is, it makes sure that the letter w_i represents a command that can follow the command represented by w_{i-1} in $\mathcal{M}$, ignoring the counter values. For example, if the command in location l_2 is $\text{INC}(x)$, then from state q_2, which is associated with l_2, we move with the letter $\text{INC}(x)$ to q_3, which is associated with l_3. The test is local, as this gadget does not check for violations involving illegal jumps due to the values of the counters. An example of the command checker for the counter machine in Fig. 2 is given in Fig. 7.

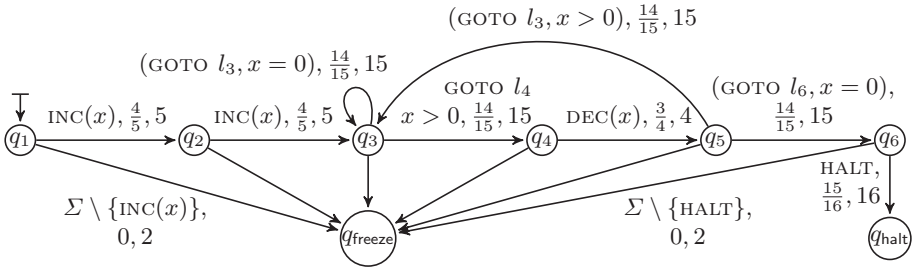


Fig. 7. The command checker that corresponds to the counter machine in Fig. 2.

The command checker, which is a DMDA, consists of states $q_1, \ldots, q_n$ that correspond to the commands $l_1, \ldots, l_n$, and the states q_{halt} and q_{freeze}. For two locations j and k, there is a transition from q_j to q_k on the letter σ iff l_k can *locally follow* l_j in a run of $\mathcal{M}$ that has σ in the corresponding location of the command trace. That is, either l_j is a GOTO l_k command (meaning $l_j = \sigma = \text{GOTO } l_k$), k is the next location after j and l_j is an INC or a DEC command (meaning $k = j + 1$ and $l_j = \sigma \in \Sigma^{\text{INCDEC}}$), l_j is an IF c=0 GOTO l_k ELSE GOTO $l_{k'}$ command with $\sigma = (\text{GOTO } l_k, c = 0)$, or l_j is an IF c=0 GOTO l_s ELSE GOTO l_k command with $\sigma = (\text{GOTO } l_k, c > 0)$. The weights and discount factors of the Σ^{NOHALT} transitions mentioned above are according to the primal functions γ_p and ρ_p respectively. For every location j such that $l_j = \text{HALT}$, there is a transition from q_j to q_{halt} labeled by the letter HALT with a weight of $\frac{15}{16}$ and a discount

factor of 16. Every other transition that was not specified above leads to q_{freeze} with weight 0 and some discount factor.

5,6. Zero-Jump Checkers. The next gadgets, depicted in Fig. 8, check for violations in conditional jumps. In this case, we use a different checker instance for each counter $c \in \{x, y\}$, ensuring that for every IF c=0 GOTO l_k ELSE GOTO $l_{k'}$ command, if the jump GOTO l_k is taken, then the value of c is indeed 0.

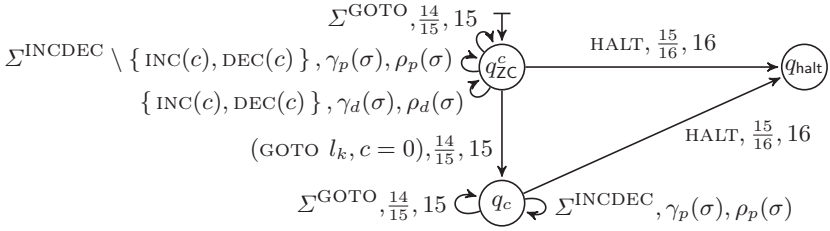


Fig. 8. The Zero-Jump Checker (for a counter $c \in \{ x, y \}$) in the NMDA $\mathcal{B}$.

Intuitively, q^c_{ZC} profits from words that have more INC(c) than DEC(c) letters, while q_c continues like $\mathcal{A}$. If the move to q_c occurred after a balanced number of INC(c) and DEC(c), as it should be in a real command trace, neither the prefix word before the move to q_c, nor the suffix word after it result in a profit. Otherwise, provided that the counter is 0 at the end of the run (as guaranteed by the negative- and positive-counters checkers), both prefix and suffix words get profits, resulting in a smaller value for the run.

7,8. Positive-Jump Checkers. These gadgets, depicted in Fig. 9, are dual to the zero-jump checkers, checking for the dual violations in conditional jumps. Similarly to the zero-jump checkers, we have a different instance for each counter $c \in \{x, y\}$, ensuring that for every IF c=0 GOTO l_k ELSE GOTO $l_{k'}$ command, if the jump GOTO $l_{k'}$ is taken, then the value of c is indeed greater than 0.

Intuitively, if the counter is 0 on a (GOTO $l_{k'}, c > 0$) command when there was no INC(c) command yet, the gadget benefits by moving from q^c_{PC0} to q_{freeze}. If there was an INC(c) command, it benefits by having the dual functions on the move from q^c_{PC0} to q^c_{PC1} over INC(c) and the primal functions on one additional self loop of q^c_{PC1} over DEC(c).

Lemma 2. *Given a two-counter machine $\mathcal{M}$, we can compute an integral DMDA $\mathcal{A}$ and an integral NMDA $\mathcal{B}$ on finite words, such that $\mathcal{M}$ 0-halts iff there exists a word $w \in \Sigma^+$ such that $\mathcal{B}(w) \geq \mathcal{A}(w)$ iff there exists a word $w \in \Sigma^+$ such that $\mathcal{B}(w) > \mathcal{A}(w)$.*

The proof uses the construction presented above, and can be found in [7].

3.2 Undecidability of Comparison

For finite words, the undecidability result directly follows from Lemma 2 and the undecidability of the 0-halting problem of counter machines [31].

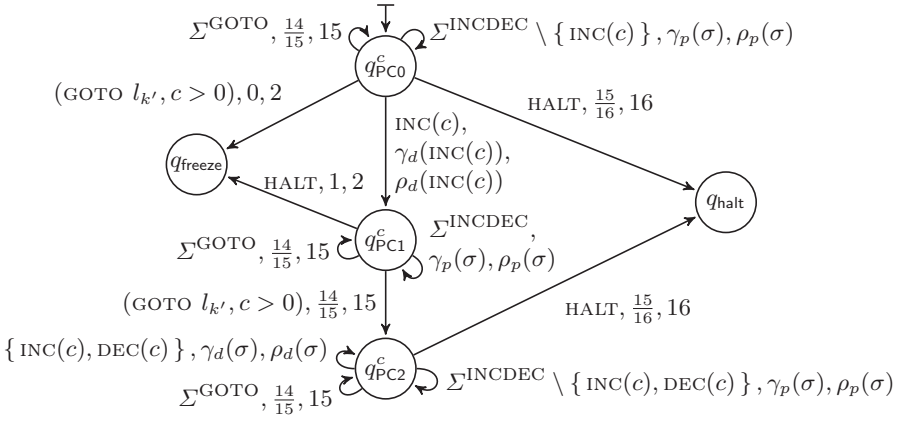


Fig. 9. The Positive-Jump Checker (for a counter c) in the NMDA $\mathcal{B}$.

Theorem 1. *Strict and non-strict containment of (integral) NMDAs on finite words are undecidable. More precisely, the problems of deciding for given integral NMDA $\mathcal{N}$ and integral DMDA $\mathcal{D}$ whether $\mathcal{N}(w) \leq \mathcal{D}(w)$ for all finite words w and whether $\mathcal{N}(w) < \mathcal{D}(w)$ for all finite words w.*

For infinite words, undecidability of non-strict containment also follows from the reduction given in Section 3.1, as the reduction considers prefixes of the word until the first HALT command. We leave open the question of whether strict containment is also undecidable for infinite words. The problem with the latter is that a HALT command might never appear in an infinite word w that incorrectly describes a halting run of the two-counter machine, in which case both automata $\mathcal{A}$ and $\mathcal{B}$ of the reduction will have the same value on w. On words w that have a HALT command but do not correctly describe a halting run of the two-counter machine we have $\mathcal{B}(w) < \mathcal{A}(w)$, and on a word w that does correctly describe a halting run we have $\mathcal{B}(w) > \mathcal{A}(w)$. Hence, the reduction only relates to whether $\mathcal{B}(w) \leq \mathcal{A}(w)$ for all words w, but not to whether $\mathcal{B}(w) < \mathcal{A}(w)$ for all words w.

Theorem 2. *Non-strict containment of (integral) NMDAs on infinite words is undecidable. More precisely, the problem of deciding for given integral NMDA $\mathcal{N}$ and integral DMDA $\mathcal{D}$ whether $\mathcal{N}(w) \leq \mathcal{D}(w)$ for all infinite words w.*

Proof. The automata $\mathcal{A}$ and $\mathcal{B}$ in the reduction given in Section 3.1 can operate as is on infinite words, ignoring the Halt-Checker gadget of $\mathcal{B}$ which is only relevant to finite words.

Since the values of both $\mathcal{A}$ and $\mathcal{B}$ on an input word w only relate to the prefix $u = \text{PREF}_{\text{HALT}(w)}$ of w until the first HALT command, we still have that $\mathcal{B}(w) > \mathcal{A}(w)$ if u correctly describes a halting run of the two-counter machine $\mathcal{M}$ and that $\mathcal{B}(w) < \mathcal{A}(w)$ if u is finite and does not correctly describe a halting run of $\mathcal{M}$.

Yet, for infinite words there is also the possibility that the word w does not contain the HALT command. In this case, the value of both $\mathcal{A}$ and the command checker of $\mathcal{B}$ will converge to 1, getting $\mathcal{A}(w) = \mathcal{B}(w)$.

Hence, if $\mathcal{M}$ 0-halts, there is a word w, such that $\mathcal{B}(w) > \mathcal{A}(w)$ and otherwise, for all words w, we have $\mathcal{B}(w) \leq \mathcal{A}(w)$. □

Observe that for NMDAs, equivalence and non-strict containment are interreducible.

Theorem 3. *Equivalence of (integral) NMDAs on finite as well as infinite words is undecidable. That is, the problem of deciding for given integral NMDAs $\mathcal{A}$ and $\mathcal{B}$ on finite or infinite words whether $\mathcal{A}(w) = \mathcal{B}(w)$ for all words w.*

Proof. Assume toward contradiction the existence of a procedure for equivalence check of $\mathcal{A}$ and $\mathcal{B}$. We can use the nondeterminism to obtain an automaton $\mathcal{C} = \mathcal{A} \cup \mathcal{B}$, having $C(w) \leq A(w)$ for all words w. We can then check whether $\mathcal{C}$ is equivalent to $\mathcal{A}$, which holds if and only if $\mathcal{A}(w) \leq \mathcal{B}(w)$ for all words w. Indeed, if $\mathcal{A}(w) \leq \mathcal{B}(w)$ then $\mathcal{A}(w) \leq \min(\mathcal{A}(w), \mathcal{B}(w)) = \mathcal{C}(w)$, while if there exists a word w, such that $\mathcal{B}(w) < \mathcal{A}(w)$, we have $\mathcal{C}(w) = \min(\mathcal{A}(w), \mathcal{B}(w)) < \mathcal{A}(w)$, implying that $\mathcal{C}$ and $\mathcal{A}$ are not equivalent. Thus, such a procedure contradicts the undecidability of non-strict containment, shown in Theorems 1 and 2. □

4 Comparison of NDAs with Different Discount Factors

We present below our algorithm for the comparison of NDAs with different discount factors. We start with automata on infinite words, and then show how to solve the case of finite words by reduction to the case of infinite words.

The algorithm is based on our main observation that, due to the difference between the discount factors, we only need to consider the combination of the automata computation trees up to some level k, after which we can consider first the best/worst continuation of the automaton with the smaller discount factor, and on top of it the worst/best continuation of the second automaton.

For an NDA $\mathcal{A}$, we define its *lowest* (resp. *highest*) *infinite run value* by $\text{LOWRUN}(\mathcal{A})$ (resp. $\text{HIGHRUN}(\mathcal{A})$) = min (resp. max) $\{\mathcal{A}(r) \mid r$ is an infinite run of $\mathcal{A}$ (on some word $w \in \Sigma^\omega)\}$.

Observe that we can use min and max (rather than inf and sup) since the infimum and supremum values are indeed attainable by specific infinite runs of the NDA (cf. [10, Proof of Theorem 9]). Notice that $\text{LOWRUN}(\mathcal{A})$ and $\text{HIGHRUN}(\mathcal{A})$ can be calculated in PTIME by a simple reduction to one-player discounted-payoff games [4].

Considering word values, we also refer to the *lowest* (resp. *highest*) *word value* of $\mathcal{A}$, defined by $\text{LOWWORD}(\mathcal{A})$ (resp. $\text{HIGHWORD}(\mathcal{A})$)= min (resp. max) $\{\mathcal{A}(w) \mid w \in \Sigma^\omega\}$. Observe that $\text{LOWWORD}(\mathcal{A}) = \text{LOWRUN}(\mathcal{A})$, $\text{HIGHWORD}(\mathcal{A}) \leq \text{HIGHRUN}(\mathcal{A})$, and for deterministic automaton, $\text{HIGHWORD}(\mathcal{A}) = \text{HIGHRUN}(\mathcal{A})$.

For an NMDA $\mathcal{A}$ with states Q, we define the *maximal difference between suffix runs* of $\mathcal{A}$ as $\text{MAXDIFF}(\mathcal{A}) = \max\{\text{HIGHRUN}(\mathcal{A}^q) - \text{LOWRUN}(\mathcal{A}^q) \mid q \in Q\}$.

Notice that $\text{MAXDIFF}(\mathcal{A}) \geq 0$ and that $\mathcal{A}^q(w)$ is bounded as follows.

$$\text{LOWRUN}(\mathcal{A}^q) \leq \mathcal{A}^q(w) \leq \text{LOWRUN}(\mathcal{A}^q) + \text{MAXDIFF}(\mathcal{A}) \tag{2}$$

Lemma 3. *There is an algorithm that computes for every input discount factors $\lambda_A, \lambda_D \in \mathbb{Q} \cap (1, \infty)$, λ_A-NDA $\mathcal{A}$ and λ_D-DDA $\mathcal{D}$ on infinite words the value of* $\min\{\mathcal{A}(w) - \mathcal{D}(w) \mid w \in \Sigma^\omega\}$.

Proof. Consider an alphabet Σ, discount factors $\lambda_A, \lambda_D \in \mathbb{Q} \cap (1, \infty)$, a λ_A-NDA $\mathcal{A} = \langle \Sigma, Q_\mathcal{A}, \iota_\mathcal{A}, \delta_\mathcal{A}, \gamma_\mathcal{A} \rangle$ and a λ_D-DDA $\mathcal{D} = \langle \Sigma, Q_\mathcal{D}, \iota_\mathcal{D}, \delta_\mathcal{D}, \gamma_\mathcal{D} \rangle$. When $\lambda_A = \lambda_D$, we can generate a λ_A-NDA $\mathcal{C} \equiv \mathcal{A} - \mathcal{D}$ over the product of $\mathcal{A}$ and $\mathcal{D}$ and compute $\text{LOWWORD}(\mathcal{C})$.

When $\lambda_A \neq \lambda_D$, **we consider first the case that** $\lambda_A < \lambda_D$.

Our algorithm unfolds the computation trees of $\mathcal{A}$ and $\mathcal{D}$, up to a level in which only the minimal-valued suffix words of $\mathcal{A}$ remain relevant – Due to the massive difference between the accumulated discount factor in $\mathcal{A}$ compared to the one in $\mathcal{D}$, any "penalty" of not continuing with a minimal-valued suffix word in $\mathcal{A}$, defined below as $m_\mathcal{A}$, cannot be compensated even by the maximal-valued word of $\mathcal{D}$, which "profit" is at most as high as $\text{MAXDIFF}(\mathcal{D})$. Hence, at that level, it is enough to look among the minimal-valued suffixes of $\mathcal{A}$ for the one that implies the highest value in $\mathcal{D}$.

For every transition $t = (q, \sigma, q') \in \delta_\mathcal{A}$, let $\text{MINVAL}(q, \sigma, q') = \gamma_\mathcal{A}(q, \sigma, q') + \frac{1}{\lambda_A} \cdot \text{LOWWORD}(\mathcal{A}^{q'})$ be the best (minimal) value that $\mathcal{A}^q$ can get by taking t as the first transition. We say that t is *preferred* if it starts a minimal-valued infinite run of $\mathcal{A}^q$, namely $\delta_{pr} = \{ t = (q, \sigma, q') \in \delta_\mathcal{A} \mid \text{MINVAL}(t) = \text{LOWWORD}(\mathcal{A}^q) \}$ is the set of preferred transitions of $\mathcal{A}$. Observe that an infinite run of $\mathcal{A}^q$ that takes only transitions from δ_{pr}, has a value equal to $\text{LOWRUN}(\mathcal{A}^q)$ (cf. [10, Proof of Theorem 9]).

If all the transitions of $\mathcal{A}$ are preferred, $\mathcal{A}$ has the same value on all words, and then $\min\{\mathcal{A}(w) - \mathcal{D}(w) \mid w \in \Sigma^\omega\} = \text{LOWRUN}(\mathcal{A}) - \text{HIGHWORD}(\mathcal{D})$. (Recall that since $\mathcal{D}$ is deterministic, we can easily compute $\text{HIGHWORD}(\mathcal{D})$.) Otherwise, let $m_\mathcal{A}$ be the minimal penalty for not taking a preferred transition in $\mathcal{A}$, meaning $m_\mathcal{A} = \min \left\{ \text{MINVAL}(t') - \text{MINVAL}(t'') \;\middle|\; \begin{array}{c} t' = (q, \sigma', q') \in \delta_\mathcal{A} \setminus \delta_{pr}, \\ t'' = (q, \sigma'', q'') \in \delta_{pr} \end{array} \right\}$. Observe that $m_\mathcal{A} > 0$.

Considering the connection between $m_\mathcal{A}$ and $\text{MAXDIFF}(\mathcal{D})$, notice first that if $\text{MAXDIFF}(\mathcal{D}) = 0$, $\mathcal{D}$ has the same value on all words, and then we have $\min\{\mathcal{A}(w) - \mathcal{D}(w) \mid w \in \Sigma^\omega\} = \text{LOWRUN}(\mathcal{A}) - \text{LOWRUN}(\mathcal{D})$. Otherwise, meaning $\text{MAXDIFF}(\mathcal{D}) > 0$, we unfold the computation trees of $\mathcal{A}$ and $\mathcal{D}$ for the first k levels, until the maximal difference between suffix runs in $\mathcal{D}$, divided by the accumulated discount factor of $\mathcal{D}$, is smaller than the minimal penalty for not taking a preferred transition in $\mathcal{A}$, divided by the accumulated discount factor of $\mathcal{A}$. Meaning, k is the minimal integer such that

$$\frac{\text{MAXDIFF}(\mathcal{D})}{{\lambda_D}^k} < \frac{m_\mathcal{A}}{{\lambda_A}^k} \tag{3}$$

Starting at level k, the penalty gained by taking a non-preferred transition of $\mathcal{A}$ cannot be compensated by a higher-valued word of $\mathcal{D}$.

At level k, we consider separately every run ψ of $\mathcal{A}$ on some prefix word u. We should look for a suffix word w, that minimizes

$$\mathcal{A}(uw) - \mathcal{D}(uw) = \mathcal{A}(\psi) + \frac{1}{{\lambda_A}^k} \cdot \mathcal{A}^{\delta_{\mathcal{A}}(\psi)}(w) - \mathcal{D}(u) - \frac{1}{{\lambda_D}^k} \cdot \mathcal{D}^{\delta_{\mathcal{D}}(u)}(w) \quad (4)$$

A central point of the algorithm is that every word that minimizes $\mathcal{A} - \mathcal{D}$ must take only preferred transitions of $\mathcal{A}$ starting at level k (full proof in [7]). As all possible remaining continuations after level k yield the same value in $\mathcal{A}$, we can choose among them the continuation that yields the highest value in $\mathcal{D}$.

Let $\mathcal{B}$ be the partial automaton with the states of $\mathcal{A}$, but only its preferred transitions δ_{pr}. (We ignore words on which $\mathcal{B}$ has no runs.) We shall use the automata product $\mathcal{B}^{\delta_{\mathcal{A}}(\psi)} \times \mathcal{D}^{\delta_{\mathcal{D}}(u)}$ to force suffix words that only take preferred transitions of $\mathcal{A}$, while calculating among them the highest value in $\mathcal{D}$.

Let $\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))} = \langle \Sigma, Q_{\mathcal{A}} \times Q_{\mathcal{D}}, \{ (\delta_{\mathcal{A}}(\psi), \delta_{\mathcal{D}}(u)) \}, \delta_{pr} \times \delta_{\mathcal{D}}, \gamma_{\mathcal{C}} \rangle$ be the partial λ_D-NDA that is generated by the product of $\mathcal{B}^{\delta_{\mathcal{A}}(\psi)}$ and $\mathcal{D}^{\delta_{\mathcal{D}}(u)}$, while only considering the weights (and discount factor) of $\mathcal{D}$, meaning $\gamma_{\mathcal{C}}((q,p),\sigma,(q',p')) = \gamma_{\mathcal{D}}(p,\sigma,p')$.

A word w has a run in $\mathcal{A}^{\delta_{\mathcal{A}}(\psi)}$ that uses only preferred transitions iff w has a run in $\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))}$. Also, observe that the nondeterminism in $\mathcal{C}$ is only related to the nondeterminism in $\mathcal{A}$, and the weight function of $\mathcal{C}$ only depends on the weights of $\mathcal{D}$, hence all the runs of $\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))}$ on the same word result in the same value, which is the value of that word in $\mathcal{D}$. Combining both observations, we get that a word w has a run in $\mathcal{A}^{\delta_{\mathcal{A}}(\psi)}$ that uses only preferred transitions iff w has a run r in $\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))}$ such that $\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))}(r) = \mathcal{D}^{\delta_{\mathcal{D}}(u)}(w)$. Hence, after taking the k-sized run ψ of $\mathcal{A}$, and under the notations defined in Eq. (4), a suffix word w that can take only preferred transitions of $\mathcal{A}$, and maximizes $\mathcal{D}^{\delta_{\mathcal{D}}(u)}(w)$, has a value of $\mathcal{D}^{\delta_{\mathcal{D}}(u)}(w) = \textsc{highrun}(\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))})$. This leads to

$$\min \{ \mathcal{A}(v) - \mathcal{D}(v) \mid v \in \Sigma^\omega \} =$$

$$\min \left\{ \mathcal{A}(\psi) + \frac{\mathcal{A}^{\delta_{\mathcal{A}}(\psi)}(w)}{{\lambda_A}^k} - \mathcal{D}(u) - \frac{\mathcal{D}^{\delta_{\mathcal{D}}(u)}(w)}{{\lambda_D}^k} \middle| \begin{array}{c} u \in \Sigma^k, w \in \Sigma^\omega, \\ \psi \text{ is a run of } \mathcal{A} \text{ on } u \end{array} \right\} =$$

$$\min_{\psi} \left\{ \mathcal{A}(\psi) + \frac{\textsc{lowrun}(\mathcal{A}^{\delta_{\mathcal{A}}(\psi)})}{{\lambda_A}^k} - \mathcal{D}(u) - \frac{\textsc{highrun}(\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))})}{{\lambda_D}^k} \middle| \begin{array}{c} u \in \Sigma^k, \\ \psi \text{ is a run} \\ \text{of } \mathcal{A} \text{ on } u \end{array} \right\}$$

and it is only left to calculate this value for every k-sized run of $\mathcal{A}$, meaning for every leaf in the computation tree of $\mathcal{A}$.

The case of $\lambda_A > \lambda_D$ is analogous, with the following changes:

- For every transition of $\mathcal{D}$, we compute $\textsc{maxval}(p,\sigma,p') = \gamma_{\mathcal{D}}(p,\sigma,p') + \frac{1}{\lambda_D} \cdot \textsc{highword}(\mathcal{D}^{p'})$, instead of $\textsc{minval}(q,\sigma,q')$.
- The preferred transitions of $\mathcal{D}$ are the ones that start a maximal-valued infinite run, that is $\delta_{pr} = \{ t = (p,\sigma',p') \in \delta_{\mathcal{D}} \mid \textsc{maxval}(t) = \textsc{highrun}(\mathcal{D}^p) \}$,

and the minimal penalty $m_{\mathcal{D}}$ is

$$m_{\mathcal{D}} = \min \left\{ \text{MAXVAL}(t'') - \text{MAXVAL}(t') \;\middle|\; \begin{array}{c} t'' = (p, \sigma'', p'') \in \delta_{pr}, \\ t' = (p, \sigma', p') \in \delta_{\mathcal{D}} \setminus \delta_{pr} \end{array} \right\}$$

- k should be the minimal integer such that $\frac{\text{MAXDIFF}(\mathcal{A})}{{\lambda_A}^k} < \frac{m_{\mathcal{D}}}{{\lambda_D}^k}$.
- We define $\mathcal{B}$ to be the restriction of $\mathcal{D}$ to its preferred transitions, and $\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))}$ as a partial λ_A-NDA on the product of $\mathcal{A}^{\delta_{\mathcal{A}}(\psi)}$ and $\mathcal{B}^{\delta_{\mathcal{D}}(u)}$ while considering the weights of $\mathcal{A}$. We then calculate $\text{LOWRUN}(\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))})$ for every k-sized run of $\mathcal{A}$, ψ, and conclude that $\min\{\mathcal{A} - \mathcal{D}\}$ is equal to $\min_{\psi} \{ \mathcal{A}(\psi) + \frac{\text{LOWRUN}(\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))})}{{\lambda_A}^k} - \mathcal{D}(u) - \frac{\text{HIGHRUN}(\mathcal{D}_{\delta_{\mathcal{D}}(u)})}{{\lambda_D}^k} \}$.
  Observe that in this case, it might not hold that all runs of $\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))}$ on the same word have the same value, but such property is not required, since we look for the minimal run value (which is the minimal word value).

□

Notice that the algorithm of Lemma 3 does not work if switching the direction of containment, namely if considering a deterministic $\mathcal{A}$ and a nondeterministic $\mathcal{D}$. The determinism of $\mathcal{D}$ is required for finding the maximal value of a valid word in $\mathcal{B}^{\delta_{\mathcal{A}}(\psi)} \times \mathcal{D}^{\delta_{\mathcal{D}}(u)}$. If $\mathcal{D}$ is not deterministic, the maximal-valued run of $\mathcal{B}^{\delta_{\mathcal{A}}(\psi)} \times \mathcal{D}^{\delta_{\mathcal{D}}(u)}$ on some word w equals the value of some run of $\mathcal{D}$ on w, but not necessarily the value of $\mathcal{D}$ on w. We also need $\mathcal{D}$ to be deterministic for computing $\text{HIGHWORD}(\mathcal{D}^p)$ in the case that $\lambda_A > \lambda_D$.

Moving to automata on finite words, we reduce the problem to the corresponding problem handled in Lemma 3, by adding to the alphabet a new letter that represents the end of the word, and making some required adjustments.

Lemma 4. *There is an algorithm that computes for every input discount factors $\lambda_A, \lambda_D \in \mathbb{Q} \cap (1, \infty)$, λ_A-NDA $\mathcal{A}$ and λ_D-DDA $\mathcal{D}$ on finite words the value of $\inf\{\mathcal{A}(u) - \mathcal{D}(u) \mid u \in \Sigma^+\}$, and determines if there exists a finite word u for which $\mathcal{A}(u) - \mathcal{D}(u)$ equals that value.*

Proof. Without loss of generality, we assume that initial states of automata have no incoming transitions. (Every automaton can be changed in linear time to an equivalent automaton with this property.)

We convert, as described below, an NDA $\mathcal{N}$ on finite words to an NDA $\hat{\mathcal{N}}$ on infinite words, such that $\hat{\mathcal{N}}$ intuitively simulates the finite runs of $\mathcal{N}$. For an alphabet Σ, a discount factor $\lambda \in \mathbb{Q} \cap (1, \infty)$, and a λ-NDA (DDA) $\mathcal{N} = \langle \Sigma, Q_{\mathcal{N}}, \iota_{\mathcal{N}}, \delta_{\mathcal{N}}, \gamma_{\mathcal{N}} \rangle$ on finite words, we define the λ-NDA (DDA) $\hat{\mathcal{N}} = \langle \hat{\Sigma}, Q_{\mathcal{N}} \cup \{q_\tau\}, \iota_{\mathcal{N}}, \delta_{\hat{\mathcal{N}}}, \gamma_{\hat{\mathcal{N}}} \rangle$ on infinite words. The new alphabet $\hat{\Sigma} = \Sigma \cup \{\tau\}$ contains a new letter $\tau \notin \Sigma$ that indicates the end of a finite word. The new state q_τ has 0-valued self loops on every letter in the alphabet, and there are 0-valued transitions from every non-initial state to q_τ on the new letter τ. Formally, $\delta_{\hat{\mathcal{N}}} = \delta_{\mathcal{N}} \cup \{(q_\tau, \sigma, q_\tau) \mid \sigma \in \hat{\Sigma}\} \cup \{(q, \tau, q_\tau) \mid q \in Q_{\mathcal{N}} \setminus \iota_{\mathcal{N}}\}$, and

$$\gamma_{\hat{\mathcal{N}}}(t) = \begin{cases} \gamma_{\mathcal{N}}(t) & t \in \delta_{\mathcal{N}} \\ 0 & \text{otherwise} \end{cases}$$

Observe that for every state $q \in Q_{\mathcal{N}}$, the following hold.

1. For every finite run $r_{\mathcal{N}}$ of $\mathcal{N}^q$, there is an infinite run $r_{\hat{\mathcal{N}}}$ of $\hat{\mathcal{N}}^q$, such that $\hat{\mathcal{N}}^q(r_{\hat{\mathcal{N}}}) = \mathcal{N}^q(r_{\mathcal{N}})$, and $r_{\hat{\mathcal{N}}}$ takes some τ transitions. ($r_{\hat{\mathcal{N}}}$ can start as $r_{\mathcal{N}}$ and then continue with only τ transitions.)
2. For every infinite run $r_{\hat{\mathcal{N}}}$ of $\hat{\mathcal{N}}^q$ that has a τ transition, there is a finite run $r_{\mathcal{N}}$ of $\mathcal{N}^q$, such that $\hat{\mathcal{N}}^q(r_{\hat{\mathcal{N}}}) = \mathcal{N}^q(r_{\mathcal{N}})$. ($r_{\mathcal{N}}$ can be the longest prefix of $r_{\hat{\mathcal{N}}}$ up to the first τ transition).
3. For every infinite run $r_{\hat{\mathcal{N}}}$ of $\hat{\mathcal{N}}^q$ that has no τ transition, there is a series of finite runs of $\mathcal{N}^q$, such that the values of the runs in $\mathcal{N}^q$ converge to $\hat{\mathcal{N}}^q(r_{\hat{\mathcal{N}}})$. (For example, the series of all prefixes of $r_{\hat{\mathcal{N}}}$).

Hence, for every $q \in Q_{\mathcal{N}}$ we have $\inf\{\, \mathcal{N}^q(r) \;\big|\; r \text{ is a run of } \mathcal{N}^q \,\} = \textsc{lowrun}(\hat{\mathcal{N}}^q)$ and $\sup\{\, \mathcal{N}^q(r) \;\big|\; r \text{ is a run of } \mathcal{N}^q \,\} = \textsc{highrun}(\hat{\mathcal{N}}^q)$. (For a non-initial state q, we also consider the "run" of $\mathcal{N}^q$ on the empty word, and define its value to be 0.) Notice that the infimum (supremum) run value of $\mathcal{N}^q$ is attained by an actual run of $\mathcal{N}^q$ iff there is an infinite run of $\hat{\mathcal{N}}^q$ that gets this value and takes a τ transition.

For every state $q \in Q_{\hat{\mathcal{N}}}$, we can determine, as follows, whether $\textsc{lowrun}(\hat{\mathcal{N}}^q)$ is attained by an infinite run taking a τ transition. We calculate $\textsc{lowrun}(\hat{\mathcal{N}}^q)$ for all states, and then start a process that iteratively marks the states of $\hat{\mathcal{N}}$, such that at the end, $q \in Q_{\hat{\mathcal{N}}}$ is marked iff $\textsc{lowrun}(\hat{\mathcal{N}}^q)$ can be achieved by a run with a τ transition. We start with q_τ as the only marked state. In each iteration we further mark every state q from which there exists a preferred transition $t = (q, \sigma, q') \in \delta_{pr}$ to some marked state q'. The process terminates when an iteration has no new states to mark. Analogously, we can determine whether $\textsc{highrun}(\hat{\mathcal{N}}^q)$ is attained by a run that goes to q_τ.

Consider discount factors $\lambda_A, \lambda_D \in \mathbb{Q} \cap (1, \infty)$, a λ_A-NDA $\mathcal{A}$ and a λ_D-DDA $\mathcal{D}$ on finite words. When $\lambda_A = \lambda_D$, similarly to Lemma 3, the algorithm finds the infimum value of $\mathcal{C} \equiv \mathcal{A} - \mathcal{D}$ using $\hat{\mathcal{C}}$, and determines if an actual finite word attains this value using the process described above.

Otherwise, the algorithm converts $\mathcal{A}$ and $\mathcal{D}$ to $\hat{\mathcal{A}}$ and $\hat{\mathcal{D}}$, and proceeds as in Lemma 3 over $\hat{\mathcal{A}}$ and $\hat{\mathcal{D}}$. According to the above observations, we have that $\inf\{\, \mathcal{A}(u) - \mathcal{D}(u) \;\big|\; u \in \Sigma^+ \,\} = \min\{\hat{\mathcal{A}}(w) - \hat{\mathcal{D}}(w) \;\big|\; w \in \Sigma^\omega\}$, and that $\inf\{\, \mathcal{A}(u) - \mathcal{D}(u) \,\}$ is attainable iff $\min\{\hat{\mathcal{A}}(w) - \hat{\mathcal{D}}(w)\}$ is attainable by some word that has a τ transition. Hence, whenever computing LOWRUN or HIGHRUN, we also perform the process described above, to determine whether this value is attainable by a run that has a τ transition. We determine that $\inf\{\, \mathcal{A}(u) - \mathcal{D}(u) \,\}$ is attainable iff exists a leaf of the computation tree that leads to it, for which the relevant values LOWRUN and HIGHRUN are attainable. □

Complexity analysis We show below that the algorithm of Lemmas 3 and 4 only needs a polynomial space, with respect to the size of the input automata, implying a PSPACE algorithm for the corresponding decision problems. We define the size of an NDA $\mathcal{N}$, denoted by $|\mathcal{N}|$, as the maximum between the number of its transitions, the maximal binary representation of any weight in it,

and the maximal unary representation of the discount factor. (Binary representation of the discount factors might cause our algorithm to use an exponential space, in case that the two factors are very close to each other.) The input NDAs may have rational weights, yet it will be more convenient to consider equivalent NDAs with integral weights that are obtained by multiplying all the weights by their common denominator [6]. (Observe that it causes the values of all words to be multiplied by this same ratio, and it keeps the same input size, up to a polynomial change.)

Before proceeding to the complexity analysis, we provide an auxiliary lemma (proof appears in [7]).

Lemma 5. *For every integers $p > q \in \mathbb{N}\setminus\{0\}$, a $\frac{p}{q}$-NDA $\mathcal{A}$ with integral weights, and a lasso run $r = t_0, t_1, \ldots, t_{x-1}, (t_x, t_{x+1}, \ldots, t_{x+y-1})^\omega$ of $\mathcal{A}$, there exists an integer b, such that $\mathcal{A}(r) = \frac{b}{p^x(p^y - q^y)}$.*

Proceeding to the complexity analysis, let the input size be $S = |\mathcal{A}| + |\mathcal{D}|$, the reduced forms of $\lambda_\mathcal{A}$ and $\lambda_\mathcal{D}$ be $\frac{p}{q}$ and $\frac{p_\mathcal{D}}{q_\mathcal{D}}$ respectively, the number of states in $\mathcal{A}$ be n, and the maximal difference between transition weights in $\mathcal{D}$ be M. Observe that $n \leq S, p \leq S, M \leq 2 \cdot 2^S$, $\frac{\lambda_\mathcal{D}}{\lambda_\mathcal{D}-1} \leq \frac{p_\mathcal{D}}{p_\mathcal{D}-q_\mathcal{D}} \leq p_\mathcal{D} \leq S$, and for $\lambda_\mathcal{D} > \lambda_\mathcal{A} > 1$, we also have $\frac{\lambda_\mathcal{D}}{\lambda_\mathcal{A}} = \frac{p \cdot q_\mathcal{D}}{q \cdot p_\mathcal{D}} \geq 1 + \frac{1}{S^2}$.

Observe that $\mathcal{A}$ has a best infinite run (and $\mathcal{D}$ has a worst infinite run), in a lasso form as in Lemma 5, with $x, y \in [1..n]$. Indeed, following preferred transitions, a run must complete a lasso, and then may forever repeat its choices of preferred transitions. Hence, $m_\mathcal{A}$, being the difference between two lasso runs, is in the form of

$$m_\mathcal{A} = \frac{b_1}{p^{x_1}(p^{y_1} - q^{y_1})} - \frac{b_2}{p^{x_2}(p^{y_2} - q^{y_2})} = \frac{b_3}{p^n(p^{y_1} - q^{y_1})(p^{y_2} - q^{y_2})} > \frac{b_3}{p^n p^{y_1} p^{y_2}}$$
$$\geq \frac{1}{p^{3n}} \geq \frac{1}{S^{3S}} \overset{\text{for } S \geq 1}{>} \frac{1}{(2^S)^{3S}} = \frac{1}{2^{3S^2}}$$

for some $x_1, x_2, y_1, y_2 \leq n$ and some integers b_1, b_2, b_3. (Similarly, we can show that $m_\mathcal{D} > \frac{1}{2^{3S^2}}$.) We have $\text{MAXDIFF}(\mathcal{D}) \leq M \cdot \frac{\lambda_\mathcal{D}}{\lambda_\mathcal{D}-1}$, hence

$$\frac{\text{MAXDIFF}(\mathcal{D})}{m_\mathcal{A}} \leq \frac{M \cdot \frac{\lambda_\mathcal{D}}{\lambda_\mathcal{D}-1}}{m_\mathcal{A}} \leq \frac{2^{1+S} \cdot S}{m_\mathcal{A}} \overset{\text{(for } S \geq 1)}{<} \frac{2^{3S}}{m_\mathcal{A}} < 2^{3S+3S^2}$$

Recall that we unfold the computation tree until level k, which is the minimal integer such that $(\frac{\lambda_D}{\lambda_A})^k > \frac{\text{MAXDIFF}(\mathcal{D})}{m_\mathcal{A}}$. Observe that for $S \geq 1$ we have $\left(\frac{\lambda_\mathcal{D}}{\lambda_\mathcal{A}}\right)^{S^2} \geq \left(1 + \frac{1}{S^2}\right)^{S^2} \geq 2$, hence for $k' = S^2 \cdot (3S + 3S^2)$, we have

$$\left(\frac{\lambda_D}{\lambda_A}\right)^{k'} = \left((\frac{\lambda_D}{\lambda_A})^{S^2}\right)^{3S+3S^2} \geq 2^{3S+3S^2} > \frac{\text{MAXDIFF}(\mathcal{D})}{m_\mathcal{A}}$$

meaning that k is polynomial in S. Similar analysis shows that k is polynomial in S also for $\lambda_\mathcal{D} < \lambda_\mathcal{A}$.

Considering decision problems that use our algorithm, due to the equivalence of NPSPACE and PSPACE, the algorithm can nondeterministically guess an optimal prefix word u of size k, letter by letter, as well as a run ψ of $\mathcal{A}$ on u, transition by transition, and then compute the value of $\mathcal{A}(\psi)+\frac{\text{LOWRUN}(\mathcal{A}^{\delta_{\mathcal{A}}(\psi)})}{{\lambda_A}^k} - \mathcal{D}(u) - \frac{\text{HIGHRUN}(\mathcal{C}^{(\delta_{\mathcal{A}}(\psi),\delta_{\mathcal{D}}(u))})}{{\lambda_D}^k}$.

Observe that along the run of the algorithm, we need to save the following information, which can be done in polynomial space:

- The automaton $\mathcal{C} \equiv \mathcal{B} \times \mathcal{D}$ (or $\mathcal{A} \times \mathcal{B}$), which requires polynomial space.
- ${\lambda_{\mathcal{A}}}^k$ (for $\mathcal{A}(\psi)$) and ${\lambda_{\mathcal{D}}}^k$ (for $\mathcal{D}(u)$). Since we save them in binary representation, we have $\log_2(\lambda^k) \leq k \log_2(S)$, requiring polynomial space.

We thus get the following complexity result.

Theorem 4. *For input discount factors $\lambda_A, \lambda_D \in \mathbb{Q} \cap (1, \infty)$, λ_A-NDA $\mathcal{A}$ and λ_D-DDA $\mathcal{D}$ on finite or infinite words, it is decidable in PSPACE whether $\mathcal{A}(w) \geq \mathcal{D}(w)$ and whether $\mathcal{A}(w) > \mathcal{D}(w)$ for all words w.*

Proof. We use Lemma 3 in the case of infinite words and Lemma 4 in the case of finite words, checking whether $\min\{\mathcal{A}(w) - \mathcal{D}(w)\} < 0$ and whether $\min\{\mathcal{A}(w) - \mathcal{D}(w)\} \leq 0$. In the case of finite words, we also use the information of whether there is an actual word that gets the desired value. □

Since integral NDAs can always be determinized [8], we get as a corollary that there is an algorithm to decide equivalence and strict and non-strict containment of integral NDAs with different (or the same) discount factors. Note, however, that it might not be in PSPACE, since determinization exponentially increases the number of states, resulting in k that is exponential in S, and storing in binary representation values in the order of λ^k might require exponential space.

Corollary 1. *There are algorithms to decide for input integral discount factors $\lambda_A, \lambda_B \in \mathbb{N}$, λ_A-NDA $\mathcal{A}$ and λ_B-NDA $\mathcal{B}$ on finite or infinite words whether or not $\mathcal{A}(w) > \mathcal{B}(w)$, $\mathcal{A}(w) \geq \mathcal{B}(w)$, or $\mathcal{A}(w) = \mathcal{B}(w)$ for all words w.*

5 Conclusions

The new decidability result, providing an algorithm for comparing discounted-sum automata with different integral discount factors, may allow to extend the usage of discounted-sum automata in formal verification, while the undecidability result strengthen the justification of restricting discounted-sum automata with multiple integral discount factors to tidy NMDAs. The new algorithm also extends the possible, more limited, usage of discounted-sum automata with rational discount factors, while further research should be put into this direction.

Acknowledgements We thank Guillermo A. Perez for stimulating discussions on the comparison of integral NDAs with different discount factors.

References

1. de Alfaro, L., Henzinger, T.A., Majumdar, R.: Discounting the future in systems theory. In: proceedings of ICALP. vol. 2719, pp. 1022–1037 (2003). https://doi.org/10.1007/3-540-45061-0_79
2. Almagor, S., Boker, U., Kupferman, O.: What's decidable about weighted automata? Information and Computatio **282** (2022). https://doi.org/10.1016/j.ic.2020.104651
3. Almagor, S., Kupferman, O., Ringert, J.O., Velner, Y.: Quantitative assume guarantee synthesis. In: proceedings of CAV. pp. 353–374. Springer (2017). https://doi.org/10.1007/978-3-319-63390-9_19
4. Andersson, D.: An improved algorithm for discounted payoff games. In: proceedings of ESSLLI Student Session. pp. 91–98 (2006)
5. Bansal, S., Chaudhuri, S., Vardi, M.Y.: Comparator automata in quantitative verification. In: proceedings of FoSSaCS. LNCS, vol. 10803, pp. 420–437 (2018). https://doi.org/10.1007/978-3-319-89366-2_23
6. Boker, U., Hefetz, G.: Discounted-sum automata with multiple discount factors. In: proceedings of CSL. LIPIcs, vol. 183, pp. 12:1–12:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.CSL.2021.12
7. Boker, U., Hefetz, G.: On the comparison of discounted-sum automata with multiple discount factors (2023). https://doi.org/10.48550/ARXIV.2301.04086
8. Boker, U., Henzinger, T.A.: Exact and approximate determinization of discounted-sum automata. Log. Methods Comput. Sci. **10**(1) (2014). https://doi.org/10.2168/LMCS-10(1:10)2014
9. Boker, U., Henzinger, T.A., Otop, J.: The target discounted-sum problem. In: proceedings of LICS. pp. 750–761 (2015). https://doi.org/10.1109/LICS.2015.74
10. Boker, U., Lehtinen, K.: History determinism vs. good for gameness in quantitative automata. In: proceedings of FSTTCS. pp. 38:1–38:20 (2021). https://doi.org/10.4230/LIPIcs.FSTTCS.2021.38
11. Brenguier, R., Clemente, L., Hunter, P., Pérez, G.A., Randour, M., Raskin, J.F., Sankur, O., Sassolas, M.: Non-zero sum games for reactive synthesis. In: Language and Automata Theory and Applications. pp. 3–23. Springer (2016)
12. Chatterjee, K., Doyen, L., Henzinger, T.A.: Alternating weighted automata. In: proceedings of FCT. LNCS, vol. 5699, pp. 3–13 (2009). https://doi.org/10.1007/978-3-642-03409-1_2
13. Chatterjee, K., Doyen, L., Henzinger, T.A.: Probabilistic weighted automata. In: proceedings of CONCUR. LNCS, vol. 5710, pp. 244–258 (2009). https://doi.org/10.1007/978-3-642-04081-8_17
14. Chatterjee, K., Doyen, L., Henzinger, T.A.: Expressiveness and closure properties for quantitative languages. Log. Methods Comput. Sci. **6**(3) (2010), http://arxiv.org/abs/1007.4018
15. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Trans. Comput. Log. **11**(4), 23:1–23:38 (2010). https://doi.org/10.1145/1805950.1805953
16. Chatterjee, K., Forejt, V., Wojtczak, D.: Multi-objective discounted reward verification in graphs and MDPs. In: proceedings of LPAR. LNCS, vol. 8312, pp. 228–242 (2013). https://doi.org/10.1007/978-3-642-45221-5_17
17. Clarke, E.M., Draghicescu, I.A., Kurshan, R.P.: A unified approach for showing language containment and equivalence between various types of ω-automata. Information Processing Letters **46**, 301–308 (1993)

18. Degorre, A., Doyen, L., Gentilini, R., Raskin, J., Toruńczyk, S.: Energy and mean-payoff games with imperfect information. In: proceedings of CSL. LNCS, vol. 6247, pp. 260–274 (2010). https://doi.org/10.1007/978-3-642-15205-4_22
19. Droste, M., Kuske, D.: Skew and infinitary formal power series. Theor. Comput. Sci. **366**(3), 199–227 (2006). https://doi.org/10.1016/j.tcs.2006.08.024
20. Filiot, E., Gentilini, R., Raskin, J.: Finite-valued weighted automata. In: proceedings of FSTTCS. LIPIcs, vol. 29, pp. 133–145 (2014). https://doi.org/10.4230/LIPIcs.FSTTCS.2014.133
21. Filiot, E., Gentilini, R., Raskin, J.: Quantitative languages defined by functional automata. Log. Methods Comput. Sci. **11**(3) (2015). https://doi.org/10.2168/LMCS-11(3:14)2015
22. Filiot, E., Löding, C., Winter, S.: Synthesis from weighted specifications with partial domains over finite words. In: proceedings of FSTTCS. pp. 46:1–46:16 (2020). https://doi.org/10.4230/LIPIcs.FSTTCS.2020.46
23. Gimbert, H., Zielonka, W.: Limits of multi-discounted markov decision processes. In: proceedings of LICS. pp. 89–98 (2007). https://doi.org/10.1109/LICS.2007.28
24. Glendinning, P., Sidorov, N.: Unique representations of real numbers in non-integer bases. Mathematical Research Letters **8**(4), 535–543 (2001)
25. Hare, K.: Beta-expansions of pisot and salem numbers. In: Waterloo Workshop in Computer Algebra (2006)
26. Hojati, R., Touati, H., Kurshan, R., Brayton, R.: Efficient ω-regular language containment. In: proceedings of CAV. LNCS, vol. 663. springer (1992)
27. Hunter, P., Pérez, G.A., Raskin, J.: Reactive synthesis without regret. Acta Informatica **54**(1), 3–39 (2017). https://doi.org/10.1007/s00236-016-0268-z
28. Kupferman, O., Vardi, M., Wolper, P.: An automata-theoretic approach to branching-time model checking. Journal of the ACM **47**(2), 312–360 (2000)
29. Madani, O., Thorup, M., Zwick, U.: Discounted deterministic markov decision processes and discounted all-pairs shortest paths. ACM Trans. Algorithms **6**(2), 33:1–33:25 (2010). https://doi.org/10.1145/1721837.1721849
30. Mahler, K.: An unsolved problem on the powers of $\frac{3}{2}$. The journal of the Australian mathematical society **8**(2), 313–321 (1968)
31. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall Series in Automatic Computation, Prentice-Hall (1967)
32. Sutton, R.S., G.Barto, A.: Introduction to Reinforcement Learning. MIT Press (1998), http://dl.acm.org/doi/book/10.5555/551283
33. Tasiran, S., Hojati, R., Brayton, R.: Language containment using non-deterministic omega-automata. In: proceedings of CHARME. LNCS, vol. 987, pp. 261–277. springer (1995)
34. Vardi, M.Y.: Verification of concurrent programs: The automata-theoretic framework. In: proceedings of LICS. pp. 167–176 (1987)
35. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency: Structure versus Automata. LNCS, vol. 1043, pp. 238–266 (1996)
36. Wang, Y., Ye, Q., Liu, T.: Beyond exponentially discounted sum: Automatic learning of return function. CoRR (2019), http://arxiv.org/abs/1905.11591
37. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theor. Comput. Sci. **158**, 343–359 (1996). https://doi.org/10.1016/0304-3975(95)00188-3

Fast Matching of Regular Patterns with Synchronizing Counting

Lukáš Holík, Juraj Síč(✉), Lenka Turoňová, and Tomáš Vojnar

Brno University of Technology, Brno, Czech Republic
{holik,sicjuraj,ituronova,vojnar}@fit.vut.cz

Abstract. Fast matching of regular expressions with *bounded repetition*, aka *counting*, such as `(ab){50,100}`, i.e., matching linear in the length of the text and independent of the repetition bounds, has been an open problem for at least two decades. We show that, for a wide class of regular expressions with counting, which we call *synchronizing*, fast matching is possible. We empirically show that the class covers nearly all counting used in usual applications of regex matching. This complexity result is based on an improvement and analysis of a recent matching algorithm that compiles regexes to deterministic counting-set automata (automata with registers that hold sets of numbers).

1 Introduction

Fast matching of regular expressions with *bounded repetition*, aka *counting*, has been an open problem for at least two decades (cf., e.g., [33]). The time complexity of the standard matching algorithms run on a regex such as `.*a.{100}` is, at best, dominated by the *length of the text multiplied by the repetition bounds*. This makes matching prone to unacceptable slowdowns since the length of the text as well as the repetition bounds are often large. In this paper, we provide a theoretical basis for matching of bounded repetition with a much more reliable performance. We show that a large and practical class of regexes with counting theoretically allows **fast matching**—in time **independent of the counter bounds** and **linear in the length of the text**.

The problem also has a strong practical motivation. Regex matching is used for searching, data validation, detection of information leakage, parsing, replacing, data scraping, syntax highlighting, etc. It is natively supported in most programming languages [6], and ubiquitous (used in 30–40 % of Java, JavaScript, and Python software [7,39,8,5]). Efficiency and predictability of regex matching is important. An extreme run-time of matching can have serious consequences, such as a failed input validation against injection attacks [41] and events like the outage of Cloudflare services [18]. Regexes vulnerabilities are also a doorway for the *ReDoS (regular expression denial of service) attack*, in which the attacker crafts a text to overwhelm a matcher (as, e.g., in the case of the outage of StackOverflow [13] or the websites exposed due to their use of the popular Express.js framework [3]). ReDoS has been widely recognized as a common and serious threat [7,9,11], with counting in regexes begin especially dangerous [37].

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 392–412, 2023.
https://doi.org/10.1007/978-3-031-30829-1_19

Matching algorithms and complexity. The potential instability of the pattern matchers is in line with the worst-case complexity of the matching algorithms. The most widely used approach to matching is backtracking (used, e.g., in standard matchers of .NET, Python, Perl, PHP, Java, JavaScript, Ruby) for its simplicity and ease of implementation of advanced features such as back-references or look-arounds. It is, however, at worst exponential to the length of the matched text and prone to ReDoS. Even though this can be improved, for instance by memoization [11], the fastest matchers used in performance critical applications all use automata-based algorithms instead of backtracking. The basis of these approaches is Thompson's algorithm [35] (also referred to as *online NFA-simulation*). Together with many optimizations, it is implemented in Intel's Hyperscan [40]. When combined with caching, it becomes the on-the-fly subset construction of a DFA, also called *online DFA-simulation* (implemented in RE2 from Google, GNU grep, SRM, or the standard matcher of Rust [17,19,30,12]). Without counting, the major factor in the worst-case complexity is $O(nm^2)$, with n being the length of the text and m the size of the number of character occurrences in the regex (m is smaller than size of the regex, the length of string defining it). We say that the *character cost*, i.e., the cost of extending the text with one character, is m^2. This is the cost of iterating through transitions of an NFA with $O(m)$ states and $O(m^2)$ transitions compiled from the regex by some classical construction [2,16,24].

Extending the syntax of regexes with *bounded quantifiers* (or *counters*), such as `(ab){50,100}`, increases the character complexity dramatically. Given k counters with the maximum bound ℓ, the number of NFA states rises to $O(m\ell^k)$, the number of transitions as well as the character cost to $O((m\ell^k)^2)$. For instance, the minimal DFA for `.*a.{k}` (i.e., a appears k characters from the end) has more than 2^k states. Moreover, note that, since k is written as a decadic numeral, its value is exponential in the size of the regex. This makes matching with already moderately high k prone to significant slowdowns and ReDoS vulnerabilities with virtually every mainstream matcher (see [36,37]). At the same time, repetition bounds easily reach thousands, in extreme tens of millions (in real-life XML [4]). Writing a dangerous counting expression is easy and it is hard to identify. Security-critical solutions may be vulnerable to counting-related ReDoS [37] despite an extra effort spent in regex design and testing, hence developers sometimes avoid counting, use workarounds and restrict functionality.

The problem of matching with bounded repetition has been addressed from the theoretical as well as from the practical perspective by a number of authors [15,4,22,26,31,20,25,36]. From these, the recent work [36] is the only one offering fast matching for a practically significant class of regexes. The algorithm of [36] compiles a regex with counting to a non-deterministic *counting automaton (CA)*, an automaton with counters that can be incremented, reset, and compared with a constant. The crux of the problem is then to convert the CA to a succinct deterministic machine that could be simulated fast in matching. The work [36] achieves this by determinizing the CA into a *counting-set automaton (CSA)*, an automaton with registers that hold *sets* of numbers. Its size is independent of the counter bounds and it updates the sets by a handful of operations that are all constant time, regardless the size of the sets. However, regexes outside the supported class do appear, the class has no syntactic characterization, and it is hard to recognize (as demonstrated also by an incorrect proposal of a syntactic

class in [36] itself). For instance, `.*a{5}` or `(ab){5}` are handled, but `.*(aa){5}` or `.*(ab){5}` are not (the requirement is technical, see Section 4).

Our contribution. In this paper, we

1. **generalize the algorithm of [36] to extend the class of handled regexes and**
2. **derive a useful syntactic characterization of the extended class.**

The derived class is characterized by *flat counting* (counting operators are not nested) where repetitions of each counted expression R are *synchronizing* (a word from R^n cannot have a prefix from R^{n+1}). It is the first clearly delimited practical class of regexes with counting that allows fast matching. It includes the easily recognizable and frequent case where every word in R has exactly one occurrence of a *marker*, a letter or a word from a finite set of markers that unambiguously identifies each occurrence of R (note that even this simple class was not handled by any previous fast algorithms, including [36]). In a our experiment with a large set of regexes from various sources, 99.6 % of non-trivial flat counting was synchronizing and 99.2 % was letter-marked.

To obtain the results (1) and (2) above, **we first modify the determinization of [36] to include the entire class of regexes with flat counting**. In a nutshell, this is achieved by two changes: (i) We allow copying and uniting of sets stored in registers, and (ii) in the determinization, we index counters of the CA by its states to handle CA in which nondeterministic runs that reach different states reach different counter values.

These modifications come with the main technical challenge that we solve in this paper: copying and uniting sets is not constant-time but linear to the size of the sets. This would make the character cost linear in the counter bound ℓ again. To remove the dependency on the counter bounds, we augment the determinization by optimizations that avoid the copying and uniting. First, to alleviate the cost of uniting, we store intersections of sets stored in registers in new shared registers, so that the intersection does not contribute to the cost of uniting the registers. Then, to increase the impact of intersection sharing, we synchronize register updates in order to make their intersections larger. We then show that if the CSA *does not replicate registers*, i.e, each register can in a transition appear on the right-hand side of only one register assignment, then it never copies registers and the cost of unions can be amortised. Finally, **we define the class of regexes with *synchronizing counting* for which the optimized CsA do not replicate counters so their simulation in matching is fast.**

Related work. In the context of regex matching, counting automata were used in several forms under several names (e.g. [20,36,4,15,31,32,33,14,23]). Besides [36] discussed above, other solutions to matching of counting regexes [15,4,22,26,31,20,25] handle small classes of regexes or do not allow matching linear in the text size and independent of counter bounds. The work [20] proposes a CA-to-CA determinization producing smaller automata than the explicit CA determinization for the limited class of monadic regexes, covered by letter-marked counting, and the size of their deterministic automata is still dependent on the counter bounds. The work [4] uses a notion of automata with counters of [15]. It focuses mostly on deterministic regexes, a class much smaller than regexes with synchronizing counting, and proposes a matching algorithm still dependent on the counter bounds. The paper [25] proposes an algorithm that takes time at

worst quadratic to the length of the text. Extended FA (XFA) of [31,32] augment NFA with a scratch memory of bits that can represent counters, and their determinization is exponential in counter bounds already for regexes such as `.*a.`$\{k\}$. The *counter-1-unambiguous* regexes of [22,23] can be directly compiled into deterministic automata called FACs, similar to our CA, independent of counter bounds, but the class is limited, excluding e.g., `.*a.`$\{k\}$.

2 Preliminaries

We use $\mathbb{N}$ to denote the natural numbers including 0. For a set S, $\mathcal{P}(S)$ denotes its powerset and $\mathcal{P}_{\text{fin}}(S)$ is the set of all *finite* subsets of S.

A *first order language (f.o.l.)* $\Gamma = (F,P)$ consists of a set of *function symbols* F and a set of *predicate symbols* P. An *interpretation* $\mathbb{I}$ of Γ with a *domain* $D_{\mathbb{I}}$ assigns a function $f^{\mathbb{I}} : D_{\mathbb{I}}^n \to D_{\mathbb{I}}$ to each n-ary $f \in F$ and a function $p^{\mathbb{I}} : D_{\mathbb{I}}^n \to \{0,1\}$ to each n-ary $p \in P$. An *assignment* of a set of variables X in $\mathbb{I}$ is a total function $\nu : X \to D_{\mathbb{I}}$. The set of *terms* $\mathsf{Terms}_{\Gamma,X}$ and the set $\mathsf{QFF}_{\Gamma,X}$ of *quantifier free formulae* (boolean combinations of atomic formulae) over Γ and X, as well as the interpretation of a term, $t^{\mathbb{I}}(\nu)$, and a formula, $\varphi^{\mathbb{I}}(\nu)$, are defined as usual. We denote by $\nu \models_{\mathbb{I}} \varphi$ that the formula φ is *satisfied* (interpreted as true) by the assignment ν. It is then *satisfiable*. We drop the sub/superscript $\mathbb{I}$ when it is clear from the context. We write $\varphi[x]$ and $t[x]$ to denote a unary formula φ or term t, respectively, with the free variable x, and we may also abuse this notation to denote the term/formula with its only free variable replaced by x. We write $t^{\mathbb{I}}(k)$ and $\varphi^{\mathbb{I}}(k)$ to denote the values $t^{\mathbb{I}}(\{x \mapsto k\})$ and $\varphi^{\mathbb{I}}(\{x \mapsto k\})$. For a set of formulae $\Psi = \{\psi_1, \ldots, \psi_n\}$, the set *Minterms*$(\Psi)$ consists of all *minterms* of Ψ, satisfiable conjunctions $\varphi_1 \wedge \cdots \wedge \varphi_n$ where for each $i : 1 \le i \le n$, φ_i is ψ_i or $\neg\psi_i$.

We fix a finite *alphabet* Σ of *symbols/letters* for the rest of the paper. Words are sequences of letters, with the *empty word* ε. The *concatenation* of words u and v is denoted $u \cdot v$, uv for short. A set of words over Σ is a *language*, the concatenation of languages is $L \cdot L' = \{u \cdot v \mid u \in L \wedge v \in L'\}$, LL' for short. *Bounded iteration* x^i, $i \in \mathbb{N}$, of a word or a language x is defined by $x^0 = \varepsilon$ for a word, $x^0 = \{\varepsilon\}$ for a language, and $x^{i+1} = x^i \cdot x$. Then $x^* = \bigcup_{i \in \mathbb{N}} x^i$. We consider a usual basic syntax of *regular expressions (regexes)*, generated by the grammar $R ::= \varepsilon \mid \mathtt{a} \mid (R) \mid RR \mid R|R \mid R^\star \mid R\{m,n\}$ where $m \in \mathbb{N}$, $n \in \mathbb{N} \cup \infty$, $0 \le m$, $0 < n$, $m \le n$, and $\mathtt{a} \in \Sigma$. We use $R\{m\}$ for $R\{m,m\}$. Regexes containing a sub-expression with the *counter* $R\{m,n\}$ or $R\{m\}$ are called *counting regexes* and m,n are *counter bounds*. We denote by $\max_R$ the maximum integer occurring in the counter bounds of regex R and we denote the number of counters by cnt_R. A regex with *flat counting* does not have nested counting, that is, in a sub-regex $S\{m,n\}$, S cannot contain counting. The *language* of a regex R is constructed inductively to the structure: $L(\varepsilon) = \{\varepsilon\}$, $L(\mathtt{a}) = \{a\}$ for $a \in \Sigma$, $L(RR') = L(R) \cdot L(R')$, $L(R^*) = L(R)^*$, $L(R|R') = L(R) \cup L(R')$, and $L(R\{m,n\}) = \bigcup_{m \le i \le n} L(R)^i$. We understand $|R|$ simply as the length of the defining string, e.g. $|\mathtt{(ab)\{10\}}| = 8$. We define $\sharp R$ as the number of character occurrences in R, formally, $\sharp a = 1$ for $a \in \Sigma$, $\sharp\varepsilon = 0$, $\sharp\mathtt{(}R\mathtt{)} = \sharp R\mathtt{\{m,n\}} = \sharp R$, and $\sharp R \cdot S = \sharp R|S = \sharp R + \sharp S$.

A *(nondeterministic) automaton (NA)* is a tuple $A = (Q, \Delta, I, F)$ where Q is a set of *states*, Δ is a set of *transitions* of the form $q \text{-}\{a\}\!\!\rightarrow r$ with $q, r \in Q$ and $a \in \Sigma$, $I \subseteq Q$ is the

set of *initial states*, and $F \subseteq Q$ is the set of *final states*. A run of A over a word $w = a_1 \dots a_n$ from state p_0 to p_n, $n \geq 0$ is a sequence of transitions $p_0 \text{-}\{a_1\}\!\!\rightarrow p_1$, $p_1 \text{-}\{a_2\}\!\!\rightarrow p_2$, …, $p_{n-1} \text{-}\{a_n\}\!\!\rightarrow p_n$ from Δ. The empty sequence is a run with $p_0 = p_n$ over ε. The run is *accepting* if $p_0 \in I$ and $p_n \in F$, and the language $L(A)$ of A is the set of all words for which A has an accepting run. A state q is *reachable* if there is a run from I to it. The *size* of the NA, $|A|$, is defined as the number of its states plus the number of its transitions. The automaton is *deterministic (DA)* iff $|I| = 1$ and for every state q and symbol a, Δ has at most one transition $q \text{-}\{a\}\!\!\rightarrow r$. The *subset construction* transforms the NA to the DA with the same language $\mathtt{DA}(A) = (Q^{\mathbb{0}}, \Delta^{\mathbb{0}}, I^{\mathbb{0}}, F^{\mathbb{0}})$ where $Q^{\mathbb{0}} \subseteq \mathcal{P}(Q)$ and $\Delta^{\mathbb{0}}$ are the smallest sets of states and transitions satisfying $I^{\mathbb{0}} = \{I\}$, $\Delta^{\mathbb{0}}$ has for each $a \in \Sigma$ and each $S \in Q^{\mathbb{0}}$ the transition $S \text{-}\{a\}\!\!\rightarrow \{s' \mid s \in S \wedge s \text{-}\{a\}\!\!\rightarrow s' \in \Delta\}$, and $F^{\mathbb{0}} = \{S \in Q^{\mathbb{0}} \mid S \cap F \neq \emptyset\}$. When the set of states Q is finite, we talk about (deterministic) *finite state* automata (NFA, DFA).[1]

This paper is concerned with the problem of fast *pattern matching*, basically a membership test: given a regex R and a text w, decide whether $w \in L(R)$. While w may be very long, R is normally small, hence the dependence on $|w|$ is the major factor in the complexity. The offline DFA simulation takes time linear in $|w|$. It (1) compiles R into an NFA $\mathtt{NFA}(R)$ (2) determinizes it, and (3) follows the DFA run over w (aka *simulates* the DFA on w), all in time and space $\Theta(2^{|\mathtt{NFA}(R)|} + |w|)$. The cost of determinization, exponential in $|\mathtt{NFA}(R)|$, is however too impractical. Modern matchers such as Grep or RE2 [19,17] therefore use the techniques of online DFA simulation, where only the part of the DFA used for processing w is constructed. It reduces the complexity to $O(\min(2^{|\mathtt{NFA}(R)|} + |w|, |w| \cdot |\mathtt{NFA}(R)|))$ (the first operand of min is the explicit determinization in case the entire DFA is constructed, plus the cost of DFA-simulation; the second operand is the cost of the online-DFA simulation, coming from that every step may incur construction of a new DFA state and transition in time $O(|\mathtt{NFA}(R)|)$). For counting regexes, the factor $|\mathtt{NFA}(R)|$ depends linearly (or more if counting is nested) on $\max_R$ and thus exponentially on $|R|$. This makes counting very problematic in practice [36,37,33]. We will present a matching algorithm which is *fast* for a specific class of regexes, meaning that its run-time is still linear in $|w|$ but is independent of $\max_R$.

3 Counting Automata

We use a rephrased definition of counting automata and counting-set automata of [36]. We will present them as a special case of a generic notion of automata with registers.

Definition 1 (Automata with registers). *An* automaton with registers *(RA) operated through an f.o.l.* Γ *under an interpretation* $\mathbb{I}$ *is a tuple* $A = (X, Q, \Delta, I, F)$ *where* X *is a set of variables called* registers*;* Q *is a finite set of* states*;* Δ *is a finite set of* transitions *of the form* $q \text{-}\{a,\varphi,u\}\!\!\rightarrow p$ *where* $p, q \in Q$, $a \in \Sigma$, $u : X \to \mathsf{Terms}_{\Gamma,X}$ *is an* update*, and* $\varphi \in \mathsf{QFF}_{\Gamma,X}$ *is a* guard*;* I *is a set of* initial configurations*, where a* configuration *is a pair of the form* $(q, \mathfrak{m})$ *where* $q \in Q$ *and* $\mathfrak{m} : X \to D_{\mathbb{I}}$ *is a register assignment called a* memory*; and* $F : Q \to \mathsf{QFF}_{\Gamma,X}$ *is a* final condition assignment.

[1] We do not require finiteness in the basic definition in order to avoid artificial restrictions of the notions of automata with registers/counters/counting sets defined later.

The language of A, $L(A)$, is defined as the language of its configuration automaton $\mathtt{Conf}(A)$. *States of* $\mathtt{Conf}(A)$ *are* configurations *of A that are reachable. I is the set of initial states of* $\mathtt{Conf}(A)$. *It has a transition* $(q,\mathfrak{m}) \dashv\{a\}\mapsto (q',\mathfrak{m}')$ *iff* $(q,\mathfrak{m})$ *is reachable and A has a transition* $\delta = q \dashv\{a,\varphi,u\}\mapsto q' \in \Delta$ *such that* $(q',\mathfrak{m}')$ *is the* image *of* $(q,\mathfrak{m})$ *under* δ, *denoted* $(q',\mathfrak{m}') = \delta(q,\mathfrak{m})$, *meaning that (1)* δ *is* enabled *in* $(q,\mathfrak{m})$, $\mathfrak{m} \models \varphi$, *and (2)* $\mathfrak{m}' = u(\mathfrak{m})$, *i.e.* $\mathfrak{m}'(x) = u(x)^{\mathbb{I}}(\mathfrak{m})$ *for each* $x \in X$. *We let* $\delta(C) = \{\delta(c) \mid c \in C\}$ *for a set of configurations C. A configuration* $(q,\mathfrak{m})$ *is a final if* $\mathfrak{m} \models F(q)$. *By* runs of *A we mean runs of* $\mathtt{Conf}(A)$. *The RA A is* deterministic *if* $\mathtt{Conf}(A)$ *is deterministic. The size of the RA is* $|A| = |Q| + \sum_{\delta\in\Delta} |\delta|$ *where* $|\delta|$ *is the sum of the sizes of the update and the guard.*

Definition 2 (Counting automata). *A* counting automaton *(CA) is an automaton with registers, called* counters, *operated through the* counting language $\Gamma_{\mathtt{cnt}}$ *that contains the unary increment function, denoted* $x+1$, *constants* 0 *and* 1, *and predicates* $x > k$ *and* $x \le k$, $k \in \mathbb{N}$, *with the standard interpretation over natural numbers, that we denote* $\mathbb{I}_{\mathtt{cnt}}$.

Regexes with counting may be translated to CA by several methods ([36,33,14,23]). We use a slightly adapted version of [14]—an extension of Glushkov's algorithm [16] to counting. For a regex R, it produces a CA $\mathtt{CA}(R) = (X, Q, \Delta, \{\alpha_0\}, F)$. Figure 1 shows an example of such CA. The construction is discussed in detail in [21], here we only overview the important properties needed in Sections 4-6:

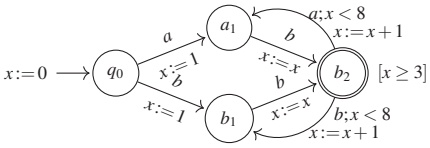


Fig. 1: $\mathtt{CA}(R)$ for $R = \mathtt{((a|b)b)\{3,8\}}$. The accepting condition of all states is $\bot$ except for b_2 whose accepting condition is written in the square brackets.

1. Every occurrence S of a counted sub-expression $T\{\min_S, \max_S\}$ of R corresponds to a unique counter x_S and a substructure A_S of $\mathtt{CA}(R)$. Outside A_S, x_S is inactive (a dead variable) and its value is 0, it is assigned 1 on entering A_S, and every iteration through A_S increments the value of x_S while reading a word from $L(T)$. Our minor modification of [14] is related to the fact that the original assigns 1 to inactive counters while we need 0.
2. $\mathtt{CA}(R)$ has at most $\sharp R + 1$ states, $cnt_R.\sharp R^2$ transitions, cnt_R counters. It has at most $\sharp R^2$ transitions if R is flat.
3. $\mathtt{CA}(R)$ has a single initial configuration $\alpha_0 = (q_0, \mathfrak{s}_0)$ s.t. $\mathfrak{s}_0(x_S) = 0$ for each $x_S \in X$.
4. Guards and final conditions are conjunctions consisting of at most one conjunct of the form $\min_S \le x_S$ or $\max_S > x_S$ per counter $x_S \in X$. A transition update may assign to $x_S \in X$ only one of the terms 0, 1, x_S, and $x_S + 1$. It has no guard on x_S if it is assigned x_S, i.e. kept unchanged, it has the guard $x_S \ge \min_S$ iff x_S is reset to 0 or 1 (a counter cannot be reset before reaching its lower bound), and it has the guard $x_S < \max_S$ iff x_S is assigned $x_S + 1$ (counter can never exceed its maximum value $\max_S$). Hence, a counter can never exceed $\max_R$.
5. Flatness of R translates to the fact that configurations of $\mathtt{CA}(R)$ assign a non-zero value to at most one counter. This implies that $\mathtt{Conf}(\mathtt{CA}(R))$ has at most $|Q|.\max_R$ states and also that $\mathtt{CA}(R)$ is *Cartesian*, a property that will be defined in Section 4 and is crucial for correctness of our CA determinization (Theorem 3 in Section 6.)

A DFA can be obtained by the subset construction in the form $\texttt{DA}(\texttt{Conf}(\texttt{CA}(R)))$, called *explicit determinization*. Due to the factor $\max_R$ in the size of $\texttt{Conf}(\texttt{CA}(R))$, the explicit determinization is exponential to $\max_R$ even if R is flat, meaning doubly exponential to $|R|$ (R has $\max_R$ written as a decadic numeral). If R is not flat, then the factor $\max_R$ is replaced by $(\max_R)^{cnt_R}$.

4 Counter-subset Construction

In this section, we formulate a modified version of determinization of CA from [36] that constructs a machine of a size independent of $\max_R$. Our version handles the entire class of Cartesian CA (defined below) and in turn also all regexes with flat counting.

The main idea of the determinization remains the same as in [36]. The standard subset construction is augmented with registers, we call them *counting sets*, that can store sets of counter values that would be generated by non-deterministic runs of the CA. The automata with counting-sets as registers are called *counting-set automata*. Our first modification of [36] is indexing of counters by states. In intuitively, this allows to handle cases such as `a*(ba|ab){5}`, where, after reading the first *ab*, the counter is either incremented or not (*b* is the first letter of the counted sub-expression or not). This would violate the uniformity property of CA necessary in [36]—the set of values generated by the non-deterministic CA runs must be the same for every CA state. In our modified version, values at distinct states are stored separately in registers indexed by those states and may differ. Then, in order to handle the indexed counters, we have to introduce a general assignment of counters, allowing to assign the *union* of other counters.[2] Intuitively, when a run non-deterministically branches into several states, each branch needs to continue with its *own copy* of the set, stored in a counter indexed by the state. The union of sets is used when the branches join again. This brings a technical challenge that we solve in this work: how to simulate the counting-set automata fast when the set union and copy are used? The solution is presented in Sections 5 and 6.

Definition 3 (Counting-set automata). *A* counting-set automaton (CSA) *is an automaton with registers operated through the* counting-set language $\Gamma_{\texttt{set}}$ *under the* number-set interpretation $\mathbb{I}^{\emptyset}_{\texttt{cnt}}$ *where the language* $\Gamma_{\texttt{set}}$ *extends the counting language* $\Gamma_{\texttt{cnt}}$ *with the constant* $\emptyset$*, binary union* $\cup$*, and set-filter functions* ∇_p *where* p *is a predicate symbol of* $\Gamma_{\texttt{cnt}}$*. For simplicity, we restrict terms assigned to counters by transition updates to the form* $t = t_1 \cup \cdots \cup t_n$ *where each* t_i *is either (a) a term of* $\Gamma_{\texttt{cnt}}$ *or* $\emptyset$*, (b) of the form* $\nabla_{p(t')}$ *where* t' *is a term of* $\Gamma_{\texttt{cnt}}$*. Each* t_i *is called an* r-term *of* t*.*

The domain of $\mathbb{I}_{\texttt{set}}$ *is* sets of natural numbers*,* $\mathcal{P}(\mathbb{N})$*. The interpretation of the predicates and functions of* $\Gamma_{\texttt{cnt}}$ *under* $\mathbb{I}_{\texttt{set}}$ *is derived from the base number interpretation of the same predicates and functions: A function returns the image of the set in the argument under the base semantics,* $f^{\mathbb{I}_{\texttt{set}}}(S) = \{f^{\mathbb{I}_{\texttt{cnt}}}(n) \mid n \in S\}$*. A set satisfies a predicate if some of its elements satisfy the base semantics of that predicate,* $p^{\mathbb{I}_{\texttt{set}}}(S) \iff \exists e \in S : p^{\mathbb{I}_{\texttt{cnt}}}(e)$*. Filters then filter out values that do not satisfy the base semantics of their predicate,* $\nabla_p^{\mathbb{I}_{\texttt{set}}}(S) = \{e \in S \mid p^{\mathbb{I}_{\texttt{cnt}}}(e)\}$*. Finally,* $\emptyset$ *is interpreted as*

[2] [36] could assign to a counter x only a constant or function of the current value of x.

the empty set and $\cup$ as the union of sets. We denote memories of the CSA by $\mathfrak{s}$ to distinguish them from memories of CA. We write DCSA to abbreviate deterministic CSA.

Less formally, registers of CSA hold sets of numbers and are manipulated by the increment $x+1$ of all values, assignment of constant sets $\{0\}$, $\{1\}$, and $\emptyset$, denoted by 0, 1, and $\emptyset$, filtering out values smaller or larger than a constant, denoted $\nabla_{x\leq k}(x)$ and $\nabla_{x<k}(x)$, and testing on a presence of a value x satisfying $x \leq k$ or $x < k$, $k \in \mathbb{N}$.

We will present an algorithm that determinizes a CA $A = (X, Q, \Delta, I, F)$, fixed for the rest of the section, into a DCSA $\mathtt{DCSA}(A) = (X^{\Downarrow}, Q^{\Downarrow}, \Delta^{\Downarrow}, I^{\Downarrow}, F^{\Downarrow})$. We assume that guards of transitions in Δ and final conditions are of the form $\bigwedge_{x\in Y} p_x[x], Y \subseteq X$, i.e. conjunctions with a at most a single atomic predicate per counter. This is satisfied by all $\mathtt{CA}(R)$, for any regex R (see the list of properties of $\mathtt{CA}(R)$ in Section 3).[3]

Runs of $\mathtt{DCSA}(A)$ will *encode* runs of $\mathtt{DA}(\mathtt{Conf}(A))$ obtained from the explicit determinization of A. Recall that the states $\mathtt{DA}(\mathtt{Conf}(A))$ are sets of configurations of A, pairs $(q, \mathfrak{m})$ of a state and a counter assignment. $\mathtt{DCSA}(A)$ will represent the sets of counter values within a DA state as run-time values of its registers.

Particularly, for every state q and a counter x of the CA, $\mathtt{DCSA}(A)$ has a register x_q in which it remembers, after reading a word w, the set of all values that x reaches in runs of the base CA on w ending in q. Hence, we have $X^{\Downarrow} = \{x_q \mid x \in X \wedge q \in Q\}$

Definition 4 (Encoding of sets of CA configurations). *A state $S = \{(q_i, \mathfrak{m}_i)\}_{i=1}^n$ of $\mathtt{DA}(\mathtt{Conf}(A))$ is encoded as the $\mathtt{DCSA}(A)$ configuration $enc(S) = (\{q_i\}_{i=1}^n, \mathfrak{s})$ where $\mathfrak{s}(x_q) = \{\mathfrak{m}_i(x) \mid q_i = q\}_{i=1}^n$.*

Since a set of assignments appearing with the state q is broken down to sets of values of the individual counters, it disregards relations between values of different counters. For instance, in the DA state $S_1 = \{(q, \{x \mapsto 0, y \mapsto 0\}), (q, \{x \mapsto 1, y \mapsto 1\})\}$, the values of x and y are either both 0 or both 1, but $enc(S_1) = (q, \{x_q \mapsto \{0,1\}, y_q \mapsto \{0,1\}\})$ does not retain this information. It is identical to the encoding of another DA state $S_2 = \{(q, \{x \mapsto 1, y \mapsto 0\}), (q, \{x \mapsto 0, y \mapsto 1\})\}$. This is the same loss of information as in the so-called Cartesian abstraction. The encoding is hence precise and unambiguous only when we assume that inside the states of $\mathtt{DA}(A)$, the relations between counters are always unrestricted—there is no information to be lost. We then call the CA *Cartesian*, as defined below. The encoding function is then unambiguous, and we call the inverse function *decoding*, denoted *dec*.

Definition 5 (Cartesian CA). *Assuming the set of counters of A is $X = \{x_i\}_{i=1}^m$, then a set C of configurations of A is* Cartesian *iff, for every state q of A, there exist sets $N_1, \ldots, N_m \subseteq \mathbb{N}$ such that $(q, \{x_i \mapsto n_i\}_{i=1}^m) \in C$ iff $(n_1, \ldots, n_m) \in N_1 \times \cdots \times N_m$. The CA A is* Cartesian *iff all states of $\mathtt{DA}(\mathtt{Conf}(A))$ are Cartesian.*

For instance, the DA states S_1 and S_2 above are not Cartesian, while $S_1 \cup S_2$ is.

Similarly as the regex to CA construction of [36], our regex to CA construction discussed in Section 3 returns a Cartesian CA when called on a flat regex.

[3] Every CA can be transformed to this form by transforming the formulae to DNF and creating clones of transitions/states for individual clauses.

Subset construction for Cartesian CA. The algorithm below is a generalization of the subset construction. Let us denote by $\mathsf{index}_q(t)$ the term that arises from t by replacing every variable $x \in X$ by x_q, analogously $\mathsf{index}_q(\varphi)$ for formulas. We have $Q^{\Uparrow} \subseteq \mathcal{P}(Q)$, the initial configuration $I^{\Uparrow} = \{enc(I)\}$, and the final conditions assign to $R \in Q^{\Uparrow}$ the disjunction of the final conditions of its elements, $F^{\Uparrow}(R) = \bigvee_{q \in R} \mathsf{index}_q(F(q))$.

We will construct $\mathtt{DCSA}(A)$ which is deterministic and its runs encode the runs of DA $\mathtt{DA}(\mathtt{Conf}(A))$. $\mathtt{Conf}(\mathtt{DCSA}(A))$ will be isomorphic to $\mathtt{DA}(\mathtt{Conf}(A))$. For that, we need for each transition δ of $\mathtt{DA}(\mathtt{Conf}(A))$ one unique transition of $\mathtt{DCSA}(A)$ over the same letter enabled in the encoding of the source of δ and generating the encoding of the target of δ. In other words, we need for each transition $dec(R,\mathfrak{s}) \text{-}\{a\}\!\!\rightarrow dec(R',\mathfrak{s}')$ of $\mathtt{DA}(\mathtt{Conf}(A))$ one unique transition $\delta' = R \text{-}\{a,\varphi,u\}\!\!\rightarrow R' \in \Delta^{\Uparrow}$ with $(R',\mathfrak{s}') = \delta'(R,\mathfrak{s})$. That transition δ' will be built by summarizing the effect of all base CA a-transitions enabled in the CA configurations of $dec(R,\mathfrak{s})$.

To construct the transition δ', we first translate each base transition $\delta = q \text{-}\{a,\varphi_\delta,u_\delta\}\!\!\rightarrow r \in \Delta$ into its set-version $\delta^{\Uparrow}$, supposed to transform an encoding of a (Cartesian) set C of configurations, $enc(C)$, into the encoding of the set of their images under δ, $enc(\delta(C))$, and enabled if δ is enabled for at least one configuration in C. To that end, assuming $\varphi_\delta = \bigwedge_{x \in X} p_x[x]$, we (1) construct the update u^{∇}_δ from u_δ by substituting in every $u_\delta(x), x \in X$ variables $y \in X$ by their filtered versions $\nabla_{p_y}(y)$, (2) add indices to registers that mark the current state, resulting in the transition $\delta^{\Uparrow} = q \text{-}\{a,\varphi^{\Uparrow}_\delta,u^{\Uparrow}_\delta\}\!\!\rightarrow r$ where $\varphi^{\Uparrow}_\delta = \mathsf{index}_q(\varphi_\delta)$ and $u^{\Uparrow}_\delta$ assigns to every $x_r, x \in X$ the term $\mathsf{index}_q(u^{\nabla}_\delta(x))$.

The states $Q^{\Uparrow}$ and the transitions $\Delta^{\Uparrow}$ are then constructed as the smallest sets satisfying that $enc(I) \in Q^{\Uparrow}$ and every $R \in Q^{\Uparrow}$ has for every $a \in \Sigma$ the outgoing transitions constructed as follows. Let $\{q_j \text{-}\{a,\varphi_j,u_j\}\!\!\rightarrow r_j\}_{j \in J}$ for some index set J be the set of *constituent a-transitions* for R, all a-transitions $\delta^{\Uparrow}$ where $\delta \in \Delta$ originates in R. To achieve determinism, $\Delta^{\Uparrow}$ has the transition $R \text{-}\{a,\psi,u\}\!\!\rightarrow R'$ for every minterm $\psi \in \mathit{Minterms}(\{\varphi_j\}_{j \in J})$. The update u and target R' are constructed from the set $\{q_j \text{-}\{a,\varphi_j,u_j\}\!\!\rightarrow r_j\}_{j \in K}$, $K \subseteq J$, of constituent transitions with guards φ_j compatible with the minterm ψ, i.e., with satisfiable $\psi \wedge \varphi_j$. R' is the set of their target states, $R' = \{r_j\}_{j \in K}$, and $u(x)$ unites all their update terms $u_j(x)$, i.e. $u(x) = \bigcup_{j \in K} u_j(x)$, for each $x \in X^{\Uparrow}$.

Example 1. When showing examples of transition updates, we write $x := t$ to denote that $u(x) = t$ and we omit the assignments $x := \emptyset$ in CSA.

Let $R = \{p,q\}$ and let the a-transitions originating at R be $q \text{-}\{a,\top,x:=x\}\!\!\rightarrow s$, $p \text{-}\{a,x<n,x:=x+1\}\!\!\rightarrow r$, and $p \text{-}\{a,x \geq m,x:=1\}\!\!\rightarrow s$. They induce three constituent transitions for R and a, $q \text{-}\{a,\top,x_s:=x_q\}\!\!\rightarrow s$, $p \text{-}\{a,x_p<n,x_r:=\nabla_{x<n}(x_p)+1\}\!\!\rightarrow r$, and $p \text{-}\{a,x_p \geq m,x_s:=1\}\!\!\rightarrow s$. A transition $R \text{-}\{a,\psi,u'\}\!\!\rightarrow R'$ is constructed for each of the following minterms ψ: $x_p<n \wedge x_p \geq m$, $\neg x_p<n \wedge x_p \geq m$, $x_p<n \wedge \neg x_p \geq m$, $\neg x_p<n \wedge \neg x_p \geq m$. For the first one, all three constituent transitions are compatible and so the update u' is $x_r := \nabla_{x<n}(x_p)+1; x_s := x_q \cup 1$ (update of x_r is taken from the first constituent transitions leading to r, update of x_s is the union of the updates of the second two transitions leading to s) and the target state is $R' = \{r,s\}$. □

$\mathtt{DCSA}(A)$ is deterministic since it has a single initial configuration and the guards of transitions originating in the same state are minterms. The size of $\mathtt{DCSA}(A)$ obviously depends only on the size of A and not on the interpretation of the language. Especially,

when A is $\mathtt{CA}(R)$ for some regex R, the size does not depend on $\max_R$. The theorem below is proved in [21].[4]

Theorem 1. $\mathtt{DCSA}(A)$ *is deterministic,* $|\mathtt{DCSA}(A)| \in O(2^{|A|})$*, and if* A *is Cartesian, then* $L(A) = L(\mathtt{DCSA}(A))$.

Since for regexes with flat counting, our regex to CA algorithm always returns a Cartesian CA, we can transform them into DCSA.

5 Fast Simulation of Counting-set Automata

In this section, we discuss how a run of a DCSA on a given word can be *simulated* efficiently to achieve fast matching. Let us fix a word $w = a_1 \cdots a_n$ together with the DCSA $A = (X, Q, \Delta, \{\alpha_0\}, F)$. We wish to construct the run of the DCSA on w and test whether the reached configuration is accepting. We aim at a running time linear to $|w|$ and independent of the sizes of the sets stored in A's registers at run-time.

We will assume that the initial configuration α_0 of A assigns to every register a singleton or the empty set. The assumption is satisfied by CSA constructed from $\mathtt{CA}(R)$, R being any regex, by the algorithms of Section 4 and also Section 6.[5]

Technically, the simulation maintains a configuration $\alpha = (q, \mathfrak{s})$, initialized with α_0, and for every i from 1 to n, it constructs the transition $\alpha \text{-}\{a_i\}\!\!\rightarrow \alpha'$ of $\mathtt{Conf}(A)$ and replaces α by the successor configuration $\alpha' = (q', \mathfrak{s}')$. We use the key ingredient of fast simulation from [36], the *offset-list data structure* for sets of numbers with constant time addition of 0/1, comparison of the maximum to a constant, reset, and increment of all values. The problem is that the newly added union and copy of sets are still linear to the size of the sets, and hence linear to the maximum counter bounds. We show how, under a condition introduced below, set copy can be avoided entirely and the cost of union can be amortized by the cost of incrementing the sets. This will again allow a CSA-simulation in time independent of $\max_A$ and falling into $O(|A| \cdot |w|)$.

First, we define a property of CSA sufficient for fast simulation—that the updates on its transitions do not *replicate counters*.

Definition 6 (Counter replication). *We say that a CSA* replicates counters *if for some transition* $q \text{-}\{a,\varphi,u\}\!\!\rightarrow r$*, some counter appears in the image of* u *twice, that is, it appears in two* r*-terms of some* $u(x)$ *or it appears in* $u(x)$ *as well as in* $u(y)$ *for* $x \neq y$*. A* non-replicating *CSA does not replicate counters.*

For instance, $\{x \mapsto x; y \mapsto x+1\}$ and $\{x \mapsto x \cup x+1, y \mapsto y\}$ are updates where x is replicated, $\{x \mapsto x+1, y \mapsto y\}$ is not a replicating update.

[4] It may be interesting to note that, as follows from our formulation of the determinization, the construction is independent of the particular f.o.l. used to manipulate registers and of its interpretation. The determinization could be applied to any kind of automata that fits the definition of automata with registers. The numbers could be manipulated by other functions and tests, natural numbers could be replaced by reals etc. The counting-set automata are themselves an instance of automata with registers. One could also think about push-down automata or, with small modifications, variants of data-word automata with registers.

[5] This is a technical assumption important in order for unions of the initial sets not to influence the overall complexity of the simulation.

Offset-list data structure. The *offset-list* data structure of [36] allows constant time implementation of the set operations of increment of all elements, reset to $\emptyset$ or $\{0\}$ or $\{1\}$, addition of 0 or 1, and comparison of the maximum with a constant.

It assigns to every counter $x \in X$ a pointer $ol(x)$ to an *offset-list pair* (o_x, l_x) with the *offset* $o_x \in \mathbb{N}$ and a sorted list $l_x = m_1, \ldots, m_k$ of integers. The data structure implementing the list needs constant access to the first and the last element, forward and backward iteration of a pointer, and insertion/deletion at/before a pointer to an element. This is satisfied for instance by a doubly-linked list that maintains pointers to the first and the last element. The offset-list pair represents the set $\mathfrak{s}(x) = \{m_1 + o_x, \ldots, m_k + o_x\}$. Union of two such sets is still linear in their size, but we will show that if the CSA does not replicate counters, the cost of set unions can be amortized by the cost of increments.

Finding the CSA transition and evaluating the update. The first step of computing α' from α is finding the transition $q \text{-}\{a_i,\varphi,u\}\!\!\rightarrow q' \in \Delta$, the only a_i-transition from q that is enabled, i.e. where $\mathfrak{s} \models \varphi$. The simplest algorithm iterates through the transitions of Δ and, for each of them, tests whether $\mathfrak{s}$ satisfies its guard. The cost of evaluating an atomic counter predicate p, i.e., deciding whether $\mathfrak{s} \models p$, is constant: since the lists l_x are sorted, we only need to access the first or the last element and the offset to decide $x < n$ or $x \geq n$, respectively. With that, the cost of evaluating φ is linear to the size of φ. The cost of the iteration through the transitions of Δ is then linear in the sum of their sizes, which is within $O(|A|)$.

Having found $q \text{-}\{a_i,\varphi,u\}\!\!\rightarrow q'$, we evaluate its update to compute $\mathfrak{s}'$ and compute α' as $(q', \mathfrak{s}')$. We will explain the algorithm and argue that the amortized cost of computing $\mathfrak{s}'$ is in $O(|X|)$. The update is evaluated by, for each $x \in X$, evaluating all r-terms in $u(x)$, uniting the results, and assigning the union to $ol(x)$.

First, we argue that evaluating an r-term t of $u(x)$, i.e. computing $t(\mathfrak{s})$, is amortized constant time. Since the counters are non-replicating, we can compute the value of each r-term $t[y]$ in situ. That is, we modify the offset-list pair (o_y, l_y) and return the pointer $ol(y)$. The original value of y can be discarded after evaluating $t[y]$ since y does not appear in any other r-term. There are 5 cases: (1) If t is 0 or 1, then we return a pointer to a fresh offset-list pair with the offset 0 and the list containing only 0 or 1, respectively. This is done in constant time.

(2) If t is $y \in Y$, then we return $ol(y)$.

(3) If t is $y + 1$, then o_y is incremented by one. This constant time implementation of the increment is the reason for pairing the lists with the offsets.

(4) If t is $\nabla_p[y]$, then l_y is filtered by the atomic predicate p. Filtering with the predicate $x \geq n$ uses the invariant of sortedness of l_y. It is done by iterating the following steps: i) test whether the list head is smaller than $n - o_y$ and ii) if yes, remove the head, if not, terminate the iteration. Every iteration is constant time: The cost of the iterations which remove an element is amortized by the cost of additions of the element to the list. What remains is only the constant cost of the last iteration which detects an element greater or equal to $n - o_y$, or that the list is empty. Filtering with $x < n$ is analogous (the iterations test and remove the last element instead of the head).

(5) If t is $\nabla_p(y) + 1$, then the construction for the constant increment is applied after the constant filter discussed above.

Next, we argue that computing the union of values of the r-terms in $u(x)$ may be amortized by the cost of evaluating the increment terms. Let $l_1, \ldots, l_n$ be the offset-list representations of the values of the terms in $u(x)$ computed by the algorithm above. The offset-list representation of their union is computed by a sequence of merging, as $merge(l_1, merge(l_2, \ldots merge(l_{n-1}, l_n)\ldots))$. Particularly, given two pointers to offset-lists l, l', $merge(l, l')$ implements their union: it chooses the offset-list that represents a set with the larger maximum, assume that it is l, and inserts the elements represented by the other list, l', to it. We say that l' *is merged into* l. This is done by the standard sorted-list merging in time $O(|l'|)$ where $|l'|$ is the length of l'. Since l' is without duplicities and with minimum 0, $O(|l'|) \subseteq O(\max(l'))$ where $\max(l')$ is the maximal element.

The $O(\max(l'))$ cost is amortized by the cost of evaluating increments. The offset-list pair at l' has seen at least $\max(l') - 1$ increments since the only elements inserted into it are 0, 1, or, during merge, elements from other sets smaller than $\max(l')$. These increments of l' are the budget used to pay for the mergeing of l' into l. After the merge, the offset-list pair of l' is discarded (as the CSA is non-replicating, it is no longer needed) hence the budget is used only once. Last, the assignment of the union to c is done by a constant time assignment of a pointer to the offset-list returned by the merge.

Overall complexity of the simulation. Let us define the cost $cost(x)$ of manipulations with the counter $x \in X$ during one step of the simulation as the sum of the costs of: (1) evaluating all r-terms containing c, (2) merging their offset-list into other ones, (3) creating offset-lists for terms 0 or 1 in $u(x)$ and merging them into other offset-lists, (4) the assignment of the result of $u(x)$ to x. The cost of processing a single letter a_i is then the sum $\sum_{x \in X} cost(x)$ and $|w| \cdot \sum_{x \in X} cost(x)$ is the cost of the entire simulation. Since the CSA is non-replicating and evaluating a single r-term is amortized constant time, the cost of (1) is in amortized constant time. The cost of (2) is amortized by increments from step (1). The creation and insertion of singletons in (3), at most two in $u(x)$, is constant time. The pointer assignment in (4) is constant time. The $cost(x)$ is therefore amortized constant time, the amortized time of evaluating the update u is in $O(|X|)$, and the cost of the updates through the simulation is in $O(|X| \cdot |w|)$. The cost of choosing the transitions, by evaluating their guards, is in $O(|A| \cdot |w|)$ by the above analysis. Analogously, the cost of testing the accepting condition at the reached configuration is in $O(|A|)$.

Theorem 2. *If A is non-replicating, then its simulation on w takes $O(|A| \cdot |w|)$ time.*

6 Augmented Determinization

In this section, we augment the subset construction from Section 4 with optimizations that prevent counter replication and hence extend the class of regexes that can be matched fast by simulation of the CSA. It optimizations are tailored to CA with the special properties of $\texttt{CA}(R)$, for a regex R, listed in Section 3.

Intuition for the optimizations. The emergence of counter replication and means of its elimination in the augmented construction, by techniques of *counter sharing* and *increment postponing*, are illustrated on simplified fragments of CA in Figure 2.

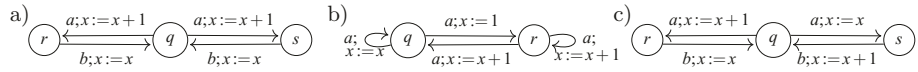


Fig. 2: Sub-structures of CA that are sources of counter replication.

In a), $\mathtt{DCSA}(\mathtt{CA}(R))$ has transitions $\{q\} \text{-}\{a,x_r{:=}x_q{+}1,x_s{:=}x_q{+}1\}\!\!\rightarrow \{r,s\} \text{-}\{b,x_q{:=}x_r{\cup}x_s\}\!\!\rightarrow \{q\}$. The first transition replicates the entire content of the x_q, the second one unites the two sets. Both transitions are expensive. The can be optimized by detecting that the values of x_s and x_r are the same, being generated by *syntactically identical* updates, and storing the values in a *shared counter* $x_{\{s,r\}}$. This would result in transitions $\{q\} \text{-}\{a,x_{\{r,s\}}{:=}x_{\{q\}}{+}1\}\!\!\rightarrow \{s,t\} \text{-}\{b,x_{\{q\}}{:=}x_{\{r,s\}}\}\!\!\rightarrow \{q\}$, with the replication and union eliminated.

Figure b) then illustrates why a counter x_P, $P \subseteq Q$, represents the set of values shared between the original counters x_p, $p \in P$. That is, x_P does not always hold the entire sets stored in the counters $x_p, p \in P$. If their values are not the same, it stores only their intersection. The value of each x_p is then partitioned among several shared counters x_S with $p \in S$. In b), $\mathtt{DCSA}(\mathtt{CA}(R))$ has transitions $q \text{-}\{a,x_q{:=}x_q;x_r{:=}1\}\!\!\rightarrow \{q,r\} \text{-}\{a,x_q{:=}x_q{\cup}x_r{+}1;x_r{:=}1{\cup}x_r{+}1\}\!\!\rightarrow \{q,r\}$, replicating the counter x_r. Counter sharing would then generate transitions $q \text{-}\{a,x_{\{q\}}{:=}x_{\{q\}};x_{\{r\}}{:=}1\}\!\!\rightarrow \{q,r\} \text{-}\{a,x_{\{q\}}{:=}x_{\{q\}};x_{\{r\}}{:=}1;x_{\{q,r\}}{:=}x_{\{r\}}{+}1\}\!\!\rightarrow \{q,r\}$ with counters $x_{\{q\}}$, $x_{\{r\}}$ for the subsets exclusive to x_q and x_r, respectively, and $x_{\{q,r\}}$ for the intersection.

Last, in c), we illustrate the technique of *increment postponing*. $\mathtt{DCSA}(\mathtt{CA}(R))$ would have transitions $\{q\} \text{-}\{a,x_r{:=}x_q{+}1,x_s{:=}x_q\}\!\!\rightarrow \{s,t\} \text{-}\{b,x_q{:=}x_r{\cup}x_s{+}1\}\!\!\rightarrow \{q\}$. Since the increments on the two branches happen in different moments, the values of x_r and x_s differ until the last increment of x_s synchronizes them. We avoid replication by storing the non-incremented value, obtained from x_q, in a counter shared by x_r and x_s and remembering that an increment of x_r has been postponed. This is marked with + in the name of the shared counter $x_{\{r^+,s\}}$. When the values of x_r and x_s synchronize (the increment is applied to x_s too), the postponed increment is evaluated and the +-mark is removed. We would create transitions $\{q\} \text{-}\{a,x_{\{r^+,s\}}{:=}x_{\{q\}}\}\!\!\rightarrow \{s,t\} \text{-}\{b,x_{\{q\}}{:=}x_{\{r^+,s\}}{+}1\}\!\!\rightarrow \{q\}$. If, before the synchronization, the value of the marked counter is either tested or incremented for the second time, we declare an *irresolvable replication* and abort the entire construction (we allow postponing of only one increment). To prevent this situation from arising needlessly, we let states remember the counters that must have the empty value and we ignore these counters.

Augmented Determinization Algorithm. The augmented determinization produces from $\mathtt{CA}(R) = (X, Q, \Delta, \{\alpha_0\}, F)$ the CSA $\mathtt{DCSA}^{\mathtt{a}}(\mathtt{CA}(R)) = (X^{\mathtt{a}}, Q^{\mathtt{a}}, \Delta^{\mathtt{a}}, \{\alpha_0^{\mathtt{a}}\}, F^{\mathtt{a}})$. Its counters in $X^{\mathtt{a}}$ are of the form x_S where $x \in X$ and $S \subseteq Q^+$ and $Q^+ = Q \cup \{q^+ \mid q \in Q\}$. The guiding principle of the algorithm is that an assignment $\mathfrak{s}^{\mathtt{a}}$ of $X^{\mathtt{a}}$ represents an assignment $\mathfrak{s}$ of the counters in $X^{\mathtt{0}}$ of $\mathtt{DCSA}(\mathtt{CA}(R))$, namely, for each $x_q \in X^{\mathtt{0}}$,

$$\mathfrak{s}(x_q) = \bigcup\nolimits_{q \in S, S \subseteq Q^+} \mathfrak{s}^{\mathtt{a}}(x_S) \cup \bigcup\nolimits_{q^+ \in S, S \subseteq Q^+} \{\, n+1 \mid n \in \mathfrak{s}^{\mathtt{a}}(x_S) \,\}. \tag{1}$$

We will use some simplifying notation. As discussed in Section 3, by the construction of $\mathtt{CA}(R)$, the increment of c and the guard $x < \max_x$ always appear on its transitions

together, without any other guard on x. Hence, in $\mathtt{DCSA}(\mathtt{CA}(R))$, all terms with an increment or filtering are of the form $\nabla_{x<\max_x}(x_{q^\circ})+1$. We will denote them by the shorthand $x_{q^\circ}\oplus 1$ (we are using q° to denote an element from the set Q^+, either q or q^+, for $q\in Q$).

The states of $\mathtt{DCSA}^{\mathtt{a}}(\mathtt{CA}(R))$ will additionally be distinguished according to which of the counters of $X^{\mathtt{a}}$ are *active*, i.e., could have a non-empty value. Counters always valued by $\emptyset$ can be ignored, which simplifies transitions and decreases the chance of an irresolvable counter replication. The states of $\mathtt{DCSA}^{\mathtt{a}}(\mathtt{CA}(R))$ are thus of the form (R,Act) where $R\subseteq Q$ and $Act\subseteq X^{\mathtt{a}}$ is a set of active counters.

The initial configuration is $\alpha_0^{\mathtt{a}}=((\{q_0\},\{x_{\{q_0\}}\mid x\in X\}),\mathfrak{s}_0^{\mathtt{a}})$ where $\mathfrak{s}_0^{\mathtt{a}}$ assigns $\{0\}$ to every $x_{\{q_0\}}, x\in X$ and $\emptyset$ to every other counter in $X^{\mathtt{a}}$. The final condition assignment $F^{\mathtt{a}}((R,Act))$ is, for each $(R,Act)\in Q^{\mathtt{a}}$, constructed from $F^{\mathbb{1}}(R)$ by replacing every predicate $p[x_q]$ by the disjunction $p[x_q]^{Act}=\bigvee_{x_S\in Act, q\in S}p[x_S]$ that encodes $p[x_q]$ using the counters of Act in the sense of (1).

The transitions in $\Delta^{\mathtt{a}}$ are constructed from transitions in $\Delta^{\mathbb{1}}$. For source state $(R,Act)\in Q^{\mathtt{a}}$, an original transition $R\text{-}\{a,\varphi,u\}\!\!\rightarrow R'\in\Delta^{\mathbb{1}}$, and set of active counters $Act\subseteq X^{\mathtt{a}}$, $\Delta^{\mathtt{a}}$ has the transition $(R,Act)\text{-}\{a,\varphi^{\mathtt{a}},u^{\mathtt{a}}\}\!\!\rightarrow(R',Act')$, constructed as follows:

The guard $\varphi^{\mathtt{a}}$ is made from φ by replacing every predicate $p[x_q]$ by the equivalent version with shared counters $p[x_q]^{Act}$ (as when constructing $F^{\mathtt{a}}$ above).

The update $u^{\mathtt{a}}$ is constructed in three steps. First, the update $u^{\mathtt{sh}}$ is made from u by expressing the r-terms of u using the shared counters $X^{\mathtt{a}}$. Each $t[x_q]$ is replaced by

$$t^{\mathtt{a}}=\bigcup\Big(\big\{\,t[x_S]\mid x_S\in Act, q\in S\,\big\}\cup\big\{\,t[x_S]\oplus 1\mid x_S\in Act, q^+\in S\,\big\}\Big)\,.$$

Notice that all postponed increments are *evaluated* in $u^{\mathtt{sh}}$, transformed to normal increments. If $u^{\mathtt{sh}}$ has an r-term $t\oplus 1\oplus 1$, i.e., a double increment, then the whole construction aborts and declares an *irresolvable counter replication*. We allow postponing only one increment.[6] Otherwise, we proceed to resolve counter replication. First, we make sure that every counter appears in the image of the update only in one kind of r-term. We collect the set *Conflict* of all r-terms $x_S\oplus 1$ of $u^{\mathtt{sh}}$ with *conflicting increments*, i.e. such that also x_S is an r-term of $u^{\mathtt{sh}}$. In update u^+, conflicting increments are *postponed*. For $x\in X$, $q\in Q$, and $u^{\mathtt{sh}}(x_q)=\bigcup T$,

$$u^+(x_q)=\bigcup\big(T\setminus Conflict\big)\ \text{ and }\ u^+(x_{q^+})=\bigcup\big\{\,x_S\mid x_S\oplus 1\in T\cap Conflict\,\big\}\,.$$

The final update $u^{\mathtt{a}}$ then resolves counter replication, by grouping r-terms replicated in u^+ under a common l-value (we call z an *l-value* of r-terms of $u^+(z)$). For an r-term t of u^+, let $\mathtt{lval}(t)$ be the set of its l-values. Note that $\mathtt{lval}(t)$ is always of the form $\{x_{q^\circ}\}_{x\in S}$ for some fixed $x\in X$ (see property 4 of $\mathtt{CA}(R)$ in Section 3). We let Act' be the set of counters x_S with $\mathtt{lval}(t)=\{x_{q^\circ}\}_{x\in S}$ for some r-term of u^+. For all $x_S\in X^{\mathtt{a}}$, if $x_S\notin Act'$ then $u^{\mathtt{a}}(x_S)=\emptyset$ else

$$u^{\mathtt{a}}(x_S)=\bigcup\big\{\,t\mid t\text{ is an r-term of }u^+\text{ and }\mathtt{lval}(t)=\{x_{q^\circ}\}_{q^\circ\in S}\,\big\}\,.$$

[6] Also transition guards and final conditions of $\mathtt{DCSA}^{\mathtt{a}}(\mathtt{CA}(R))$ must not contain the +-mark since evaluating them regardless the postponed increments would return incorrect results. However, declaring counter replication on seeing a double increment here covers these cases due to the structural properties of $\mathtt{CA}(R)$.

Example 2. Let us have $R \text{-}\{a,\varphi,u\}\!\!\rightarrow R' \in \Delta^{\natural}$ created in Example 1 with $R = \{p,q\}$, $R' = \{r,s\}$, $\varphi = x_p{<}n \wedge x_p{\geq}m$, and $u = \{x_r := x_p \oplus 1, x_s := x_q \cup 1\}$. Let $Act = \{x_{\{p,q\}}, x_{\{p,q^+\}}\}$. Then $u^{\mathtt{sh}} = \{x_r := x_{\{p,q^+\}} \oplus 1 \cup x_{\{p,q\}} \oplus 1, x_s := x_{\{p,q^+\}} \oplus 1 \cup x_{\{p,q\}} \cup 1\}$. Note that the x_q in $u(x_s)$ becomes $x_{\{p,q^+\}} \oplus 1$, corresponding to the right part of the definition of $t^{\mathtt{a}}$ (the postponed increment x_{q^+} is evaluated in $u^{\mathtt{sh}}$). Note that the r-term $x_{\{p,q\}} \oplus 1$ is in *Conflict* as $x_{\{p,q\}}$ is an r-term of $u^{\mathtt{sh}}$ too. Therefore it is postponed in u^+, i.e. $u^{\mathtt{sh}}(x_r) = x_{\{p,q\}} \oplus 1 \cup \cdots$ becomes $u^+(x_{r^+}) = x_{\{p,q\}}$. We get $u^+ = \{x_r := x_{\{p,q^+\}} \oplus 1, x_s := x_{\{p,q^+\}} \oplus 1 \cup x_{\{p,q\}} \cup 1, x_{r^+} := x_{\{p,q\}}\}$. Finally, $u^{\mathtt{a}}$ groups r-terms replicated in u^+ under a common l-value: $u^{\mathtt{a}} = \{x_{\{r,s\}} := x_{\{p,q^+\}} \oplus 1, x_{\{s\}} := 1, x_{\{s,r^+\}} := x_{\{p,q\}}\}$. The next active counters are $Act' = \{x_{\{r,s\}}, x_{\{s\}}, x_{\{s,r^+\}}\}$. Note that, for $x_{\{p,q^+\}}$, the postponed increment at p^+ was synchronized on this transition, while the conflict at $x_{\{p,q\}}$ was solved by postponing increment and marking r with $^+$. □

The algorithm either returns the CSA $\mathtt{DCSA}^{\mathtt{a}}(\mathtt{CA}(A))$, or detects an irresolvable counter replication, in which case $\mathtt{DCSA}^{\mathtt{a}}(\mathtt{CA}(A))$ does not exist.[7] Let $m = \sharp R$ and recall that n denotes the length of the matched text, $|w|$. Since $\mathtt{CA}(R)$ has at most m states and m^2 transitions, a basic analysis of the algorithm's data structures reveals that the resulting CSA has at most 2^{2^m} states, each with at most 2^{m^2} outgoing transitions, each transition of the size in $O(m2^m)$. Because $\mathtt{DCSA}^{\mathtt{a}}(\mathtt{CA}(A))$ encodes $\mathtt{DCSA}(\mathtt{CA}(A))$, it has the same language, and it also inherits its determinism. Since it does not replicate counters, it can be simulated in pattern matching fast, in time linear to the text and independent of the counter bounds. The following theorem is proved in [21].

Theorem 3. *For R with flat counting, if $\mathtt{DCSA}^{\mathtt{a}}(\mathtt{CA}(R))$ exists, then it does not replicate counters, its size is in $O(2^{2^m}m)$, $L(\mathtt{CA}(R)) = L(\mathtt{DCSA}^{\mathtt{a}}(\mathtt{CA}(R)))$, and it can be simulated on a word w of the length n in time $O(2^{2^m}mn)$.*

Matching can be done in time of constructing the CSA plus its simulation, which in the sum is indeed fast, not dependent on k and linear in n. It can also be noted that the m in the exponents above is not the size of the entire regex, but only the size of the counted sub-regexes.

7 Regexes with Synchronizing Counting

Finally, in this section we define the class of regexes with synchronizing counting, which precisely captures when the CSA created by our construction in Section 6 does not replicate counters and hence allow fast matching (in the sense of Theorem 3).

Definition 7 (Regexes with synchronizing counting). *A regex has* synchronizing counting *iff it has no sub-expression $S\{n,m\}$ where for some $k \in \mathbb{N}$, a word from $L(S)^k$ has a prefix from $L(S)^{k+1}$.*

For instance, `(ac*){1,4}(ab|ba){3,5}(a(ab)*){2,8}` is a regex with synchronizing counting as each word from $L(\mathtt{ac*})^k$ must contain the symbol a exactly k times,

[7] Aborting the construction here simplifies the description, but it would also be possible to continue the construction and return a DCSA that does not guarantee fast simulation.

words from $L(\texttt{ab|ba})^k$ must have exactly $2k$ symbols, and words from $L(\texttt{a(ab)*})^k$ can be uniquely split at the first a in the $\texttt{a(ab)*}$. In comparison, $\texttt{(a|aa)\{2,5\}}$ does not have synchronizing counting as $a \cdot a \cdot a$ is a prefix of $aa \cdot aa$.

Intuitively, there is no pair of paths through $\texttt{CA}(S\{\texttt{m,n}\})$ starting at the same state, over the same word, ending in the same state, where the number of increments differs by two. In such case, $\texttt{DCSA}^{\texttt{a}}(\texttt{CA}(S\{\texttt{m,n}\}))$ would have to delay two increments, which our construction does not allow. The theorem below is proved in [21].

Theorem 4. *Given a regex R with flat counting, the algorithm of Section 6 returns $\texttt{DCSA}^{\texttt{a}}(\texttt{CA}(R))$ if and only if R has synchronizing counting.*

Corollary 1. *Regexes with flat synchronizing counting have a fast matching algorithm.*

Proof. From Theorems 3 and 4.

Counting with Markers. Even though designing and recognizing synchronizing counting is usually intuitive, it may also be tricky. For instance, $\texttt{(\\\\d+\\\\.)\{3\}}$, from the database of real-world regexes we use in our experiment, has synchronizing counting, while $\texttt{ICE\_Dims.\{92\}((\_?(X|\textbackslash d+))\{13\})}$ does not.[8] A vast majority of real-world regexes we examined fortunately belong to very easily recognizable subclasses of synchronizing counting. The most wide-spread and easy to recognize are regexes with *letter-marked counting*, where every sub-expression $S\{\texttt{m,n}\}$ has a set of marker letters such that every word from $L(S)$ has exactly one occurrence of a marker letter. [9]

Marker letters may be generalized to *marker words*, though, markers that can arise by concatenation of several words from $L(S)$ cannot be used. The condition that has to be satisfied is that any word from $L(S)^k$, $k \in \mathbb{N}$, has exactly k non-overlapping occurrences of marker words as infixes. Another sufficient property of S is that it has words of a *uniform length*. The idea of markers may be generalized further until the point when the set of marker words is specified by general regexes, when we get precisely the synchronizing counting. The regexes with letter-marked counting are easily human as well as machine recognizable (see a simple $O(|R|^2)$-time algorithm in [21]).

8 Practical Considerations

Although the main point of this work is the theoretical feasibility of fast matching with synchronizing counting, we will also argue that the results are of practical relevance. To this end, we show experimentally that synchronizing counting and marked counting cover a majority of practical regexes. We also give arguments that matching with the CSA constructed in Section 6 can be done efficiently.

[8] An automated way of identifying synchronizing counting would be running the CSA-to-DCSA determinization from Section 6, but this is exponential to $|R|$.

[9] That letter-marked counting is a strict superset of the class that is in [36] conjectured as handled by the algorithm of [36]. The conjecture of [36] is also not correct, as shown in [21].

8.1 Occurrence of Synchronizing Counting in Practice

To substantiate the practical relevance of synchronizing counting regexes, we examined a large sample of practical regexes using a simple checker of letter-marked counting. The benchmark consists of over 540 000 regexes collected from (1) a large scale analysis of software projects [10]; (2) regexes used by network intrusion detection systems Snort [27], Bro [29], Sagan [34], and the academic papers [42,38]; (4) the RegExLib database of regexes [28].

From the regexes that we could parse[10], 31 975 contained counting. We selected those with flat counting and with the sum of upper bounds of counters larger than 20 (as was done in [36] to filter out counting with small bounds that can be handled through counter unfolding and traditional methods)[11]. This left us with 5 751 regexes. From these, only 46 regexes (0.8 %) have counting that is not letter-marked. Furthermore, we manually checked these regexes and we identified that 22 of them have synchronizing counting. We have therefore found only 24 regexes with non-synchronizing counting, i.e., 0.4 % of the examined set of regexes with flat counting.

The 24 non-synchronizing regexes are listed in [21]. Some of them may clearly be rewritten with synchronizing counting, such as `(.+){25}(.*)`, which can be rewritten as `.{25,}(.*)`. We speculate that some of them might in fact represent a mistake, such as `(.*){1,32000}[bc]` where the counter matches the empty word, or `(\n\s+)(criterion .*\n)(\s.+){1,99}` where the `\s.+` might have been intended as `\s\S+` (`\s` are white spaces, `\S` are all the other characters). Synchronizing counting seems to capture the intuition with which counting is often written, hence reporting non-synchronizing counting might help identifying bugs.

By the same methodology and from a nearly identical benchmark, [36] arrived to a sample of 5 000 regexes with flat counting with the sum of bounds larger than 20. The algorithm of [36] did not cover 571 regexes from the 5 000, which is 11 % of the examined set of regexes with flat counting (in contrast to the 0.4 % with non-synchronizing counting and the 0.8 % with counting that is not letter-marked, measured on a slightly larger set of regexes). The two sets of regexes with flat counting, the 5 751 of ours and the 5 000 of [36], are not perfectly identical, however. Differences are to a small degree caused by differences in the base database ([36] uses about 18 more regexes that are proprietary and excludes 26 regexes with counter bounds larger than 1 000), and to a larger degree by small differences in the parsers.

8.2 Practical Efficiency of Matching with Synchronizing Counting

The size and the worst-case time of simulation of $\mathtt{DCSA}^{\mathtt{a}}(\mathtt{CA}(R))$ are still exponential to the number of states of $\mathtt{CA}(R)$ (namely, $O(2^{2^m}m)$ and $O(2^{2m}mn)$ where $m = \sharp R$ equals the number of states of $\mathtt{CA}(R)$, cf. Theorem 3). The potential problem is that the algorithm may generate at most 2^m counters, and this potentially threatens practicality of our matching algorithm.

[10] We did not parse 38 558 regexes since their syntax was broken or contained some advanced features we do not support.

[11] 926 regexes contain nested counting and 25297 regexes contain small upper bounds.

First, it should be noted that the m in the exponent can be decreased from the size of the entire regex to the size of the counted sub-expression, which is usually very small. Then, although an efficient implementation is beyond the scope of this paper and we are leaving it as a future work, we give some indirect arguments for practicality of the CA-to-CSA algorithm.[12]

By the standard techniques of register allocation [1], it is possible to decrease the number of counters and counter assignments other than identity dramatically. In fact, simply eliminating needless renaming of counters and reusing the same name whenever possible, our algorithm creates CSA isomorphic to those of [36] when run on regexes handled by [36]. The work [36] already shows that simulating these CSA may be done efficiently and that it brings dramatic improvements over best matchers on counting-intensive examples.

In our experience with hand-simulating the algorithm on practical examples, cases not handled by [36] do not behave much differently, and the numbers of CSA counters do not have a strong tendency to explode.

9 Conclusions

We have extended the regex matching algorithm of [36] and shown that the extended version allows fast pattern matching of so-called synchronising regexes, a class of regexes that we have newly introduced. The class of synchronising regexes significantly extends all previously known classes of regexes that allow fast matching and covers a majority of regexes appearing in practice (wrt. our empirical study).

In the future, we plan to study extensions of the presented techniques to regexes with nested counting (non-flat). This will probably require a more sophisticated alternative of the offset-list data structure for sets, capable of storing relations of numbers. An interesting question is also how and when regexes can be rewritten to a synchronizing form and for what cost.

Acknowledgment

This work has been supported by the Czech Ministry of Education, Youth and Sports project LL1908 of the ERC.CZ programme, the Czech Science Foundation project 23-06506S, and the FIT BUT internal project FIT-S-23-8151.

References

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition). Addison Wesley (August 2006), http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0321486811

[12] A competitive matcher that runs on real-world regexes requires an extensive infrastructure, optimized data structures for the shared registers, and ideally an on-the-fly version of the CA-to-CSA determinization (similar to the online DFA simulation).

2. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theoretical Computer Science **155**(2), 291 – 319 (1996). https://doi.org/10.1016/0304-3975(95)00182-4, https://doi.org/10.1016/0304-3975(95)00182-4
3. Baldwin, A.: Regular expression denial of service affecting express.js. https://medium.com/node-security/regular- expression-denial-of-service-affecting-express-js-9c397c164c43 (2016)
4. Björklund, H., Martens, W., Timm, T.: Efficient incremental evaluation of succinct regular expressions. In: CIKM'15. ACM (2015). https://doi.org/10.1145/2806416.2806434
5. Chapman, C., Stolee, K.T.: Exploring regular expression usage and context in python. In: Zeller, A., Roychoudhury, A. (eds.) Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016. pp. 282–293. ACM (2016). https://doi.org/10.1145/2931037.2931073, https://doi.org/10.1145/2931037.2931073
6. contributors, W.: Regular expression—wikipedia (2019), https://en.wikipedia.org/w/index.php?title=Regular_expression&%20oldid=852858998
7. Davis, J.C.: Rethinking regex engines to address ReDoS. In: ESEC/FSE'19. pp. 1256–1258. ACM (2019)
8. Davis, J.C., Coghlan, C.A., Servant, F., Lee, D.: The impact of regular expression denial of service (redos) in practice: an empirical study at the ecosystem scale. In: Leavens, G.T., Garcia, A., Pasareanu, C.S. (eds.) Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018. pp. 246–256. ACM (2018). https://doi.org/10.1145/3236024.3236027, https://doi.org/10.1145/3236024.3236027
9. Davis, J.C., Coghlan, C.A., Servant, F., Lee, D.: The impact of regular expression denial of service (ReDoS) in practice: An empirical study at the ecosystem scale. In: ESEC/FSE'18. pp. 246–256. ACM (2018)
10. Davis, J.C., Michael IV, L.G., Coghlan, C.A., Servant, F., Lee, D.: Why aren't regular expressions a lingua franca? An empirical study on the re-use and portability of regular expressions. In: ESEC/FSE'19. pp. 1256–1258. ACM (2019)
11. Davis, J.C., Servant, F., Lee, D.: Using selective memoization to defeat regular expression denial of service (ReDoS). In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. pp. 1–17. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00032, https://doi.org/10.1109/SP40001.2021.00032
12. docs.rs: regex - rust. https://docs.rs/regex/1.5.4/regex/ (2021)
13. Exchange, S.: Outage postmortem. http://stackstatus.net/post/147710624694/outage-postmortem-july-20-2016 (2016)
14. Gelade, W., Gyssens, M., Martens, W.: Regular expressions with counting: Weak versus strong determinism. In: Mathematical Foundations of Computer Science 2009. pp. 369–381. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03816-7_32
15. Gelade, W., Gyssens, M., Martens, W.: Regular expressions with counting: Weak versus strong determinism. SIAM J. Comput. **41**(1), 160–190 (2012). https://doi.org/10.1137/100814196, extended version of paper in MFCS'09
16. Glushkov, V.M.: The abstract theory of automata. Russian Math. Surveys **16**, 1–53 (1961). https://doi.org/10.1070/RM1961v016n05ABEH004112
17. Google: RE2. https://github.com/google/re2
18. Graham-Cumming, J.: Details of the Cloudflare outage on july 2, 2019. https://blog.cloudflare.com/details-of-the-cloudflare-outage-on-july-2-2019/ (2019)
19. Haertel, M., et al.: GNU grep. https://www.gnu.org/software/grep/

20. Holík, L., Lengál, O., Saarikivi, O., Turoňová, L., Veanes, M., Vojnar, T.: Succinct determinisation of counting automata via sphere construction. In: Proc. of APLAS'19. LNCS, vol. 11893, pp. 468–489. Springer (2019). https://doi.org/10.1007/978-3-030-34175-6_24
21. Holík, L., Síč, J., Turoňová, L., Vojnar, T.: Fast matching of regular patterns with synchronizing counting (technical report). Tech. rep., Brno University of Technology (2023), https://doi.org/10.48550/arXiv.2301.12851
22. Hovland, D.: Regular expressions with numerical constraints and automata with counters. In: ICTAC. LNCS, vol. 5684, pp. 231–245. Springer (2009). https://doi.org/10.1007/978-3-642-03466-4_15
23. Hovland, D.: The membership problem for regular expressions with unordered concatenation and numerical constraints. In: Language and Automata Theory and Applications. pp. 313–324. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28332-1_27
24. Hromkovič, J., Seibert, S., Wilke, T.: Translating regular expressions into small ε-free nondeterministic finite automata. In: Reischuk, R., Morvan, M. (eds.) STACS 97. pp. 55–66. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)
25. Kilpeläinen, P., Tuhkanen, R.: Regular expressions with numerical occurrence indicators - preliminary results. In: SPLST'03. pp. 163–173. University of Kuopio, Department of Computer Science (2003)
26. Kilpeläinen, P., Tuhkanen, R.: One-unambiguity of regular expressions with numeric occurrence indicators. Information and Computation **205**(6), 890–916 (2007). https://doi.org/10.1016/j.ic.2006.12.003
27. M. Roesch et al.: Snort: A Network Intrusion Detection and Prevention System,. http://www.snort.org
28. RegExLib.com: The Internet's first Regular Expression Library. http://regexlib.com/
29. Robin Sommer et al.: The Bro Network Security Monitor, http://www.bro.org
30. Saarikivi, O., Veanes, M., Wan, T., Xu, E.: Symbolic regex matcher. In: Vojnar, T., Zhang, L. (eds.) TACAS'2019. LNCS, vol. 11427, pp. 372–378. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_24, https://doi.org/10.1007/978-3-030-17462-0_24
31. Smith, R., Estan, C., Jha, S.: XFA: faster signature matching with extended automata. In: IEEE Symposium on Security and Privacy. IEEE (2008). https://doi.org/10.1109/SP.2008.14
32. Smith, R., Estan, C., Jha, S., Siahaan, I.: Fast signature matching using extended finite automaton (XFA). In: ICISS'08. LNCS, vol. 5352, pp. 158–172. Springer (2008). https://doi.org/10.1007/978-3-540-89862-7_15
33. Sperberg-McQueen, M.: Notes on finite state automata with counters. https://www.w3.org/XML/2004/05/msm-cfa.html, https://www.w3.org/XML/2004/05/msm-cfa.html, accessed: 2018-08-08
34. The Sagan team: The Sagan Log Analysis Engine, https://quadrantsec.com/sagan_log_analysis_engine/
35. Thompson, K.: Programming techniques: Regular expression search algorithm. Commun. ACM **11**(6), 419–422 (1968)
36. Turoňová, L., Holík, L., Lengál, O., Saarikivi, O., Veanes, M., Vojnar, T.: Regex matching with counting-set automata. Proc. ACM Program. Lang. **4**(OOPSLA), 218:1–218:30 (2020)
37. Turoňová, L., Holík, L., Lengál, O., Veanes, M., Vojnar, T.: Counting in regexes considered harmful (2022)
38. Češka, M., Havlena, V., Holík, L., Lengál, O., Vojnar, T.: Approximate reduction of finite automata for high-speed network intrusion detection. In: Proc. of TACAS'18. LNCS, vol. 10806. Springer (2018). https://doi.org/10.1007/978-3-319-89963-3_9

39. Wang, P., Stolee, K.T.: How well are regular expressions tested in the wild? In: Leavens, G.T., Garcia, A., Pasareanu, C.S. (eds.) Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018. pp. 668–678. ACM (2018). https://doi.org/10.1145/3236024.3236072, https://doi.org/10.1145/3236024.3236072
40. Wang, X., Hong, Y., Chang, H., Park, K., Langdale, G., Hu, J., Zhu, H.: Hyperscan: A fast multi-pattern regex matcher for modern CPUs. In: 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). pp. 631–648. USENIX Association, Boston, MA (Feb 2019), https://www.usenix.org/conference/nsdi19/presentation/wang-xiang
41. Wübbeling, M.: Regular expression security. ADMIN **55** (2020)
42. Yang, L., Karim, R., Ganapathy, V., Smith, R.: Improving NFA-based signature matching using ordered binary decision diagrams. In: Recent Advances in Intrusion Detection. pp. 58–78. Springer Berlin Heidelberg (2010)

Compositional Learning for Interleaving Parallel Automata

Faezeh Labbaf[1] (✉), Jan Friso Groote[2], Hossein Hojjat[1,3], and Mohammad Reza Mousavi[4]

[1] Tehran Institute for Advanced Studies (TeIAS), Khatam University, Tehran, Iran
f.labaf@khatam.ac.ir
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
j.f.Groote@tue.nl
[3] University of Tehran, Tehran, Iran
hojjat@ut.ac.ir
[4] King's College London, London, UK
mohammad.mousavi@kcl.ac.uk

Abstract. Active automata learning has been a successful technique to learn the behaviour of state-based systems by interacting with them through queries. In this paper, we develop a compositional algorithm for active automata learning in which systems comprising interleaving parallel components are learned compositionally. Our algorithm automatically learns the structure of systems while learning the behaviour of the components. We prove that our approach is sound and that it learns a maximal set of interleaving parallel components. We empirically evaluate the effectiveness of our approach and show that our approach requires significantly fewer numbers of input symbols and resets while learning systems. Our empirical evaluation is based on a large number of subject systems obtained from a case study in the automotive domain.

1 Introduction

Active automata learning has been successfully used to learn models of complex industrial systems such as communication- and security protocols [11], biometric passports [2], smart cards [1], large-scale printing machines [33], and lithography machines for integrated circuits [32,15]; we refer to the recent survey by Howar and Steffen on the practical applications of active automata learning [16]. Throughout these applications of automata learning, scalability issues have been pointed out [32,15]. It has also been suggested that compositional learning, i.e., learning a system through learning its components, is a promising approach to tame the complexity of learning [10,12].

Some early attempts have been recently made in learning structured models of systems [27,10] (we refer to the Related Work for an in-depth analysis). For example, the approach proposed by al-Duhaiby and Groote [10] decomposes the learning process into learning its parallel components; however, it relies on a deep knowledge of the system under learning, and the intricate interaction

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 413–435, 2023.
https://doi.org/10.1007/978-3-031-30829-1_20

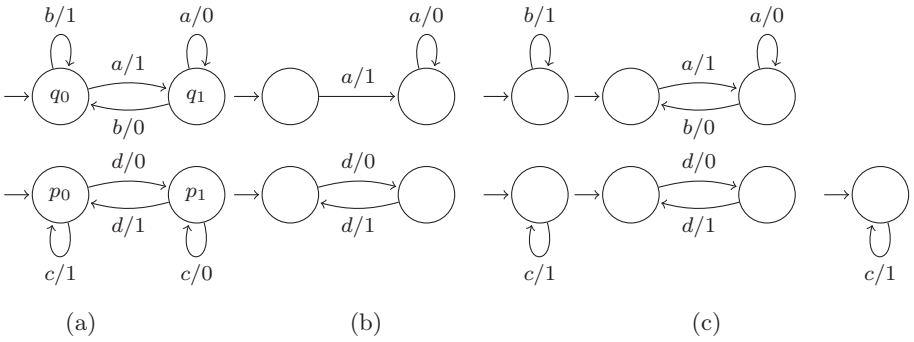


Fig. 1: (a) Initial system with two concurrent FSMs (b) Partition the input alphabet to 4 elements and learn each component individually (c) Use the counter-example ab to merge two components

of the various actions being learned. In this paper, we propose an approach based on Dana Angluin's celebrated L* algorithm [6], to learn the components of a system featuring an interleaving parallel composition. Our approach, called CL*, does not assume any pre-knowledge of the structure and the alphabet of these components; instead, we learn this information automatically and on-the-fly, while providing a rigorous guarantee of the learned information. This is particularly relevant in the context of legacy and black-box systems where architectural discovery is challenging [8,22].

The gist of our approach is to learn the System Under Learning (SUL) in separate components with disjoint alphabets. We start with a partition comprising only singleton sets. The interleaving parallel composition of the components gives us the total behavior of the system. We pass the result to the teacher, and by exploiting the counter-examples returned, we iteratively merge the alphabet of the individual components.

Example. Figure 1(a) shows an example of two parallel Finite State Machines (FSMs) over the input alphabet $\{a, b, c, d\}$ and output alphabet $\{0, 1\}$. We start by partitioning the alphabet into disjoint singleton sets of elements. The parallel composition of the 4 learned FSMs of Figure 1(b) does not comply with the original system, and the teacher may return the counter-example ab. The string ab generates the output sequence 10 in (a) but the output sequence in (b) is 11. The counter-example suggests to merge the sets $\{a\}$ and $\{b\}$ and restart the learning process which leads to the FSMs in Figure 1(c). One further merging step results in learning the original system. We provide a theoretical proof of correctness of this compositional construction, meaning that it is guaranteed to construct a correct system.

To study the effectiveness of our approach in practice, we designed an empirical experiment to investigate the following two research questions:

RQ1 Does CL* require fewer resets, compared to L*?
RQ2 Does CL* require fewer input symbols, compared to L*?

Our research questions are motivated by the following facts: 1) Resets are a major contributing factor in learning practical systems as they are immensely time- and resource consuming [31]. Hence, reducing the number of resets can have a significant impact in the learning process. 2) The total number of symbols used in interacting with the system under learning provides us with a total measure of cost for the learning process and hence, reducing the total cost is a fair indicator of improved efficiency [36,9].

To answer these questions, we use a benchmark based an industrial automotive system. We design a number of experiments on learning various combinations of components in this system, gather empirical data, and analyse them through statistical hypothesis testing. Our results indicate that our compositional approach significantly improves the efficiency of learning compared to the monolithic L* learning algorithm. The implementation of the algorithm, experiments, and their results can be found on-line in our lab package [23] (`https://github.com/faezeh-lbf/CL-Star`).

The remainder of this paper is organised as follows. In Section 2, we review the related work and position our research with respect to the state of the art. In Section 3, we present the preliminary definitions that are used throughout the rest of the paper. In Section 4, we present our algorithm and its proof of correctness and termination. We evaluate our algorithm on a benchmark from the automotive domain in Section 5. We conclude the paper and present the directions of our ongoing and future research in Section 6.

2 Related Work

Active automata learning is a technique used to find the underlying model of a black box system by posing queries and building a hypothesis in an iterative manner. There is substantial early work in this domain, e.g., under the name system identification or grammar inference; we refer to the accessible introduction by Vaandrager [36] for more information. A seminal work in this domain is the L* algorithm by Dana Angluin [6], which comes with theoretical complexity bounds for the learning process using a representation called the "Minimally Adequate Teacher" (MAT).

MAT hypothesises a teacher that is capable of responding to membership queries (MQs) and equivalence queries (EQs); the former checks the outcome of a sequence of inputs (e.g., with their respective outputs, or with their membership in the language of the automaton) and the latter checks whether a hypothesised automaton is equivalent to the system under learning. Our work replaces a single MAT with multiple MATs that can potentially run in parallel and learn different components of the black-box system automatically.

Learning structured systems and in particular, compositional learning of parallel systems has been studied recently in the literature. Moerman [27] proposes

an algorithm to learn parallel interleaving Moore machines. Our algorithm differs from Moerman's algorithm in that in the parallel composition of Moore machines, the output of each individual component is explicitly specified, because the output of the system is specified as a tuple of the outputs of its components. In other words, the underlying structure is immediately exposed by considering the type of outputs produced by the system under learning. However, in our approach, we need to identify the components and assign outputs to them on-the-fly since the decomposition is not explicit in parallel composition. Al-Duhaiby and Groote [10] learn parallel labelled transitions systems with the possibility of synchronisation among them. In order to develop their algorithm, they assume a priori knowledge of mutual dependencies among actions in terms of a confluence relation. This type of information is difficult to obtain and the domain knowledge in this regard may be error prone. Particularly for legacy and large black-box systems (e.g., binary code), architectural discovery has proven challenging [8,22]. We address this challenge and go beyond the existing approaches by learning about confluence of actions on-the-fly through observing the minimal counter-examples generated by the MAT(s).

Frohme and Steffen [12] introduce a compositional learning approach for Systems of Procedural Automata [13]; these are collections of DFAs that may "call" each, akin to the way non-terminals may be used in defining other non-terminals in a grammar. Their approach is essentially different from ours in that the calls across automata are assumed to be observable and hence the general structure is assumed to be known; in our approach, we learn the structure by observing implicit dependencies among the learned automata through analysing counter-examples. Also their approach is aimed at a richer and more expressive type of systems, namely pushdown systems, which justifies the requirement for additional information.

L^* has been improved significantly in the past few years; the major improvements upon L^* can be broadly categorised into three categories: 1) improving the data structures used to store and retrieve the learned information [21,31,19,37]; 2) improving the way counter-examples are processed in refining the hypothesis [31,28,3,17]; 3) learning more expressive models, such as register- [18,14] and timed automata [34,5]. This third category of improvements is orthogonal to our contribution and extension of our approach can be considered in those contexts as well.

Two notable recent improvements, in the first two categories, are $L^\#$ [37] and L^λ [17], respectively. $L^\#$ uses the notion of *apartness* to organise and maintain a tree-shaped data-structure about the learned automaton. L^λ uses a search-based method to incorporate the information about the counter-example into the learned hypothesis. The improvements brought about by L^λ can be readily incorporated into our approach, particularly since our approach relies on finding minimal counter-examples. Integrating our approach into $L^\#$ requires a more careful consideration of maintaining and composing tree-shaped data structures when detecting dependencies. We expect that both of these combinations will further improve the efficiency of our proposed method.

3 Preliminaries

In this section, we review the basic notions used throughout the remainder of the paper. We start by formalising the notion of a finite state machine, which is the underlying model of the system under learning and move on to parallel composition and decomposition (called projection) as well as the concept of (in)dependent actions, which are essential in identifying the parallel components. Finally, we conclude this section by recalling the basic concepts of active automata learning and the L^* algorithm.

3.1 Finite State Machines (FSMs)

Finite state machines (also called Mealy machines), defined below, are straightforward generalisations of finite automata in which the transitions produce outputs (rather than only indicating acceptance or non-acceptance):

Definition 1. *(Finite State Machine) A Finite State Machine (FSM) M is a sixtuple $(S, s_0, I, O, \delta, \lambda)$ where :*

- *S is a finite set of internal states,*
- *$s_0 \in S$ is the initial state,*
- *I is a set of actions, representing the input alphabet,*
- *O is the set of outputs,*
- *$\delta : S \times I \to S$ is a total state transition function,*
- *$\lambda : S \times I \to O$ is a total output function.*

An FSM starts in the initial state s_0 and accepts a word (a sequence of actions of its input alphabet) in order to produce an equally-sized sequence of outputs. State transition- δ and output function λ determine the next state and the output of an FSM upon receiving a single input. For each $s, s' \in S$, $i \in I$, and $o \in O$, we write $s \xrightarrow{i/o} s'$ when $\delta(s, i) = s'$ and $\lambda(s, i) = o$.

State transitions are extended inductively from a single input $i \in I$, to a sequence of inputs $w \in I^*$, i.e., we define $\delta(s, \epsilon) = s$ and $\lambda(s, \epsilon) = \epsilon$ where ϵ is the empty sequence; and for $s \in S, w \in I^*$, and $a \in I$, we have $\delta(s, wa) = \delta(\delta(s, wa), a)$ and $\lambda(s, wa) = \lambda(s, w)\lambda(\delta(s, w), a)$, where juxtaposition of sequences denotes concatenation. For the sake of conciseness, we write $\delta(w)$ and $\lambda(w)$ instead of $\delta(s_0, w)$ and $\lambda(s_0, w)$.

In much of the literature in active learning, the system under learning is assumed to be complete and deterministic and we follow this common assumption in Definition 1 by requiring the state transition and output relations to be total functions. While the determinism assumption is essential for our forthcoming results to hold, we expect that the existing recipes for learning non-deterministic state machines can be made compositional using a similar approach as ours.

3.2 (De)Composing FSMs

Our aim is to produce a compositional learning algorithm for systems composed of interleaving parallel components, defined below. Due to the interleaving nature of parallel composition and determinism of the system under learning, the alphabets of these components are assumed to be disjoint.

Definition 2. *(Interleaving Parallel Composition) For two FSMs* $M_i = (S_i, s_{0_i}, I_i, O_i, \delta_i, \lambda_i)$*, with* $i \in \{0, 1\}$*, where* $I_0 \cap I_1 = \emptyset$*, the interleaving parallel composition of* M_0 *and* M_1*, denoted by* $M_0 \mid\mid M_1$*, is an FSM defined as*

$$(S_0 \times S_1, (s_{0_0}, s_{0_1}), I_0 \cup I_1, O_0 \cup O_1, \delta, \lambda)$$

where δ *and* λ *are defined by*

$$\delta((s_0, s_1), a) = \begin{cases} (\delta_0(s_0, a), s_1) \text{ if } a \in I_0, \\ (s_0, \delta_1(s_1, a)) \text{ otherwise, and} \end{cases} \qquad \lambda((s_0, s_1), a) = \begin{cases} \lambda_0(s_0, a) \text{ if } a \in I_0, \\ \lambda_1(s_1, a) \text{ otherwise.} \end{cases}$$

For $s_0 \in S_0$*,* $s_1 \in S_1$*, and* $a \in I_0 \cup I_1$

Next, we define the notions of projections for FSMs and for words; these notions are further used in the notion of (in)dependence and eventually in our proof of correctness to establish that the composed system has the same behaviour as the composition of the learned components.

Definition 3. *(Projection of an FSM) The projection of an FSM* $M = (S, s_0, I, O, \delta, \lambda)$ *on a set of inputs* $I' \subseteq I$ *denoted by* $P(M, I')$*, is an FSM* $(S, s_0, I', O', \delta', \lambda')$*, where*

- $\delta'(s, a) = \delta(s, a)$ *for* $a \in I'$*,*
- $\lambda'(s, a) = \lambda(s, a)$ *for* $a \in I'$*, and*
- $O' = \{o \in O \mid \exists a \in I'.\ \exists s \in S.\ \lambda(s, a) = o\}$.

Definition 4. *(Projection of a word) The projection of a word* $w \in I^*$ *on a set of inputs* $I' \subseteq I$*, denoted by* $P_{I'}(w)$*, is inductively defined as follows:*

$$P_{I'}(\epsilon) := \epsilon,$$
$$P_{I'}(au) := \begin{cases} aP_{I'}(u) \text{ if } a \in I', \\ P_{I'}(u) \quad \text{otherwise.} \end{cases}$$

Definition 5. *(Projection of an output sequence) The projection of the output sequence* $w = o_1 \dots o_n$ *with respect to an equally-sized sequence of inputs* $v = i_1, \dots, i_n \in I^*$ *and a subset of inputs* $I' \subseteq I$*, denoted by* $P_{I'}(w, v)$*, is defined as follows:*

$$P_{I'}(\epsilon, \epsilon) := \epsilon,$$
$$P_{I'}(ow, av) := \begin{cases} oP_{I'}(w, v) \text{ if } a \in I', \\ P_{I'}(w, v) \quad \text{otherwise.} \end{cases}$$

Definition 6. *((In)Dependent Actions) Consider an FSM M with a set of inputs I. The subsets $I_0, \ldots, I_n \subseteq I$ form an independent partition of I when for any $u \in I^*$, $\lambda_{P(M,I_0)||\ldots||P(M,I_n)}(u) = \lambda_M(u)$. Two inputs $i_0, i_1 \in I$ are independent when they belong to two distinct subsets of an independent partition. Two input actions are dependent, when they are not independent.*

Example. The partition $\big\{\{a\},\{b\},\{c,d\}\big\}$ in Figure 1(a) is not an independent partition because $\lambda_M(ab) = 10$ but $\lambda_{P(M,\{a\})||P(M,\{b\})||P(M,\{c,d\})}(ab) = 11$.

It immediately follows from Definition 6 and associativity of parallel composition (with respect to trace equivalence) that any coarser partitioning based on an independent partition is also an independent partitioning; this is formalised in the following corollary.

Corollary 1. *By combining two or more sets of an independent partition, the resulting partition remains independent.*

Moreover, it holds that any smaller subset of an independent partitioning is also an independent partitioning of the original state machine projected on the alphabet of the smaller subset, as specified and proven below.

Lemma 1. *Consider an independent partition $I_0, \ldots, I_n$ of inputs I for an FSM M; then for $K \subseteq \{0, \ldots, n\}$, $\{I_i \mid i \in K\}$ is an independent partition for $P(M, \bigcup_{i\in K}(I_i))$.*

Proof. Consider any subset $K \subseteq \{0, \ldots, n\}$ and $\{I_i \mid i \in K\}$ and consider any input sequence $u \in (\bigcup_{i\in K} I_i)^*$. Since u does not contain a symbol that is in any I_j for $j \notin K$, we have that $\lambda_{||_{i\in K}P(M,I_i)}(u) = \lambda_{P(M,I_0)||\ldots||P(M,I_n)}(u)$. Since $I_0, \ldots, I_n$ are independent, it follows likewise that $\lambda_{P(M,I_0)||\ldots||P(M,I_n)}(u) = \lambda_M(u)$. Using again that u has no symbol in any I_j for $j \notin K$, we know that $\lambda_M(u) = \lambda_{P(M,\bigcup_{i\in K}(I_i))}(u)$. Hence, $\lambda_{||_{i\in K}P(M,I_i)}(u) = \lambda_{P(M,\bigcup_{i\in K}(I_i))}(u)$, which was to be shown.

■

Lemma 2. *For any independent partition $I_0, \ldots, I_n \subseteq I$, $w \in I^*$ and $0 \leq i \leq n$, and state s it holds that $P_{I_i}(\lambda_M(s,w), w) = \lambda_{P(M,I_i)}(s, P_{I_i}(w))$.*

Proof. The proof uses induction on the length of w. Instead of proving the thesis, we prove the following stronger statement, which is possible because M can be viewed as the parallel construction of independent components.

$$P_{I_i}(\lambda_M((s_0, \ldots s_n), w), w) = \lambda_{P(M,I_i)}((s'_0, \ldots, s'_n), P_{I_i}(w)) \text{ with } s_i = s'_i.$$

Note that the lemma directly follows from this. Below we write $\boldsymbol{s}$ for $s_0, \ldots, s_n$, and likewise for $\boldsymbol{s'}$ and $\boldsymbol{s''}$.

The base case ($|w| = 0$) holds trivially as $w = \epsilon$. For the induction step we assume that the induction hypothesis holds for $|w| = k$ and we show that it holds for $w' = aw$ for arbitrary $a \in I$.

We first consider the case where $a \notin I_i$. We derive

$$\begin{aligned} P_{I_i}(\lambda_M(\boldsymbol{s}, aw), aw) &= P_{I_i}(\lambda_M(\boldsymbol{s}, a)\lambda_M(\delta(\boldsymbol{s}, a), w), aw) && \text{Definition 1} \\ &= P_{I_i}(\lambda_M(\delta(\boldsymbol{s}, a), w), w) && \text{Definition 5.} \\ &= \lambda_{P(M,I_i)}(\boldsymbol{s}', P_{I_i}(w)) && \text{Induction hypothesis.} \\ &= \lambda_{P(M,I_i)}(\boldsymbol{s}'', P_{I_i}(aw)) && \text{Definition 4.} \end{aligned}$$

By construction the i-th state in $\delta(\boldsymbol{s}, a)$ is equal to s_i as $a \notin I_i$. Hence, using the induction hypothesis, $\boldsymbol{s}'_i = s_i$. By definition $\boldsymbol{s}' = \delta(\boldsymbol{s}'', a)$ and hence, $\boldsymbol{s}''_i = \boldsymbol{s}'_i = \boldsymbol{s}_i$ as we had to show.

The other case we must consider is $a \in I_i$. Again the derivation is straightforward.

$$\begin{aligned} P_{I_i}(\lambda_M(\boldsymbol{s}, aw), aw) &= P_{I_i}(\lambda_M(\boldsymbol{s}, a)\lambda_M(\delta(\boldsymbol{s}, a), w), aw) && \text{Definition 1} \\ &= \lambda_M(\boldsymbol{s}, a)P_{I_i}(\lambda_M(\delta(\boldsymbol{s}, a), w), w) && \text{Definition 5.} \\ &= \lambda_M(\boldsymbol{s}', a)\lambda_{P(M,I_i)}(\delta(\boldsymbol{s}', a), P_{I_i}(w)) && \text{Induction hypothesis.} \\ &= \lambda_{P(M,I_i)}(\boldsymbol{s}, P_{I_i}(aw)) && \text{Definition 4.} \end{aligned}$$

Using the induction hypothesis it follows that $s_i = s'_i$, which concludes the proof. ■

3.3 Model Learning

Active model learning, introduced by Dana Angluin, was originally designed to formulate a hypothesis $\mathscr{H}$ about the behavior of a System Under Learning (SUL) as an FSM. Model learning is often described in terms of the Minimally Adequate Teacher (MAT). In the MAT framework, there are two phases: (i) hypothesis construction, where a learning algorithm poses Membership Queries (MQ) to gain knowledge about the SUL using reset operations and input sequences; and (ii) hypothesis validation, where based on the model learned so far, the learner proposes a hypothesis $\mathscr{H}$ about the "language" of the SUL and asks Equivalence Queries (EQ) to test it. The results of the queries are organised in an observation table. The table is iteratively refined and is used to formulate $\mathscr{H}$.

Definition 7. *(Observation Table) An observation table is a triple (S, E, T), where $S \subseteq I^*$ is a prefix-closed set of input strings (i.e., prefixes); $E \subseteq I^+$ is a suffix-closed set of input strings (i.e., suffixes); and T is a table where rows are labeled by elements from $S \cup (S.I)$, columns are labeled by elements from E, such that for all $pre \in S \cup (S.I)$ and $suf \in E$, $T(pre, suf)$ is the SUL's output suffix of size $|suf|$ for the input sequence $pre.suf$.*

The L* algorithm initially starts with S only containing the empty word ϵ, and E equals set of inputs alphabet I. Two crucial properties of the observation table, closedness and consistency, defined below, allow for the construction of a hypothesis.

Definition 8. *(Closedness Property) An observation table is closed iff for all $w \in S.I$ there is a $w' \in S$ that for all $suf \in E$, $T(w, suf) = T(w', suf)$ holds.*

Definition 9. *(Consistency Property) An observation table is consistent iff for all $pre_1, pre_2 \in S$, if for all $suf \in E$, $T(pre_1, suf) = T(pre_2, suf)$, it holds that $T(pre_1.\alpha, suf) = T(pre_2.\alpha, suf)$ for all $\alpha \in I$, $suf \in E$.*

MQs are posed until these two properties hold, and once they do, a hypothesis $\mathscr{H}$ is formulated. After formulating $\mathscr{H}$, L* works under the assumption that an EQ can return either a counter-example (CE) exposing the non-conformance, or yes, if $\mathscr{H}$ is indeed equivalent to the SUL. When a `CE` is found, a `CE` processing method adds prefixes and/or suffixes to the observation table and hence refines $\mathscr{H}$. The aforementioned steps are repeated until EQ confirms that $\mathscr{H}$ and SUL are the same. In between MQs, we often need to bring the FSM back to a known state; this is done through reset operations, which are one of our metrics for measuring the efficiency of the algorithm. EQs are posed by running a large number of test-cases and hence they are (two- to three) orders of magnitude larger than MQs. These test cases are generated through a random-walk of the graph or through a deterministic algorithm that tests all states and transitions for a given fault model. Two examples of deterministic test-case generation algorithms are the W- and WP-method [7]. It appears from recent empirical evaluations that for realistic systems deterministic equivalence queries are not efficient [4].

Since we are going to be learning the system in terms of components with disjoint alphabets, we define the following projection operator that removes all the transitions that are not in the projected alphabet. Our compositional learning algorithm basically learns a black-box with respect to its projection on the actions available in each purported component.

Definition 10. *(L^* with projected alphabet) Given an SUL $M = (S, s_0, I, O, \delta, \lambda)$ and $I' \subset I$, $L^*(M, I')$ returns $P(M, I')$ by running algorithm L^* with projected alphabet I' on M.*

4 Compositional Active Learning

In this section, we present an algorithm that learns the SUL in separate components and uses the interleaving parallel composition of the learned components to reach the total behavior of the system. Each component has an input alphabet I_i, which is disjoint from the alphabet of all the other components. The set of the input alphabets of components $I^F = \{I_1, \ldots, I_n\}$ is a partition of the total system's input alphabet. The main idea is to find an independent partitioning I^F. To reach such a partitioning, we start with a partition with singleton sets and iteratively merge those sets that are found to be dependent on each other. Then for $I_i \in I^F$, we learn the SUL with the projected alphabet I_i, and compute the product of the obtained components with interleaving parallel composition. The result is equivalent to the SUL if I^F is an independent partition.

Algorithm 1: Compositional Learning Algorithm (CL*)

```
   Result: ℋ
1  Input: I^F = {I_1, ..., I_n}, M
2  ℋ ← LearnInParts(M, I^F)
3  eq ← Equivalence-Query(ℋ, M)
4  while eq ≠ yes do
5  |  CE ← eq
6  |  D ← InvolvedSets(CE, I^F)
7  |  I^F ← Composition(I^F, D)
8  |  ℋ ← LearnInParts(M, I^F)
9  |  eq ← Equivalence-Query(ℋ, M)
10 end
11 return ℋ, I^F
```

Definition 11. *(LearnInParts) The LearnInParts function gets* $M = (S, s_0, I, O, \delta, \lambda)$ *and the partition* $I^F = \{I_1, \ldots, I_n\}$ *of* I *and returns the interleaving parallel composition of the learned components.*

$$LearnInParts(M, I^F) = L^*(M, I_1) \ || \ldots || \ L^*(M, I_n).$$

Definition 12. *(Composition) Given a partition* $I^F = \{I_1, \ldots, I_n\}$ *and* $D \subseteq \{1, \ldots, n\}$*, the Composition of* I^F *over* D *merges all the* I_i *(*$i \in D$*) in* I^F*.*

$$Composition(I^F, D) = (I^F \setminus \{I_i | i \in D\}) \cup \{ \bigcup_{i \in D} I_i \}.$$

Example. If $I^F = \{\{a\}, \{b\}, \{c\}, \{d\}\}$ and $D = \{1, 3, 4\}$, then $Composition(I^F, D) = \{\{a, c, d\}, \{b\}\}$.

Definition 13. *(InvolvedSets) The function InvolvedSets gets a counter-example* CE *and a partition* $I^F = \{I_1, \ldots, I_n\}$ *and returns indices of the sets in* I^F *that contains at least one character of* CE*:*

$$InvolvedSets(\texttt{CE}, I^F) = \{j \mid I_j \in I^F, \ \exists i \ \texttt{CE}[i] \in I_j\},$$

where the i^{th} *character of* CE *is denoted by by* $\texttt{CE}[i]$*.*

The function *InvolvedSets* allows us to detect some dependent sets by using a minimal counter-example since all actions in the counter-example are dependent, as we prove in Theorem 2.

Algorithm 1 shows the pseudo-code of the compositional learning algorithm. Initially the algorithm is called with the singleton partitioning I^F of the alphabet I and the SUL M, i.e., if the input alphabet is $I = \{a_1, a_2, \ldots, a_n\}$, then the initial partition of the alphabet will be $I^F = \{\{a_1\}, \{a_2\}, \ldots, \{a_n\}\}$. The *LearnInParts* method on line 2 learns each of the components given the corresponding alphabet set using the algorithm L* and returns the interleaving

parallel composition of the learned components. If the oracle (MAT) returns yes for the equivalence query regarding hypothesis $\mathscr{H}$, the algorithm terminates and returns $\mathscr{H}$. Otherwise an(other) iteration of the loop is performed. The *InvolvedSets* method in line 6 extracts the dependent sets from the counter-example returned by the oracle; subsequently, *Composition* merges those sets into one. The *LearnInParts* method in line 8 is run again and the loop continues until the correct hypothesis is learned. We assume that the oracle always returns a minimal counter-example; this assumption is used in the proof of soundness (Theorem 2).

4.1 Termination Analysis

To prove the termination of our algorithm, we start with the following lemma which indicates how the counter-example is used to merge the partitions.

Lemma 3. *Let $I^F = \{I_1, \ldots, I_m\}$ be a partition of the system's input alphabet. If the teacher responds with a counter-example* `CE`*, then there are at least two actions $u \in I_i, v \in I_j$ in* `CE` *such that $I_i \neq I_j \ \wedge \ I_i, I_j \in I^F$.*

Proof. We prove this by contradiction. Suppose `CE` consists of actions that all belong to I_i. Let $C_i = L^*(M, I_i)$ with output function λ_{C_i}. Since the output of L^* is always the correctly learned FSM of the SUL, $\lambda_M(\texttt{CE}) = \lambda_{C_i}(\texttt{CE})$. Also, since C_i is a component of $\mathscr{H}$ produced by *LearnInParts*, $\lambda_{\mathscr{H}}(\texttt{CE}) = \lambda_{C_i}(\texttt{CE})$ based on Definition 2. This means `CE` can not be a counter-example. ■

The next lemma uses Lemma 3 to show how counter-examples will ensure progress in the algorithm, eventually guaranteeing termination.

Lemma 4. *At each round of the algorithm CL*, $|I^F|$ decreases by at least 1.*

Proof. By Lemma 3, at each round of the algorithm, at least two dependent sets are found by InvolvedSets, and the algorithm merges these dependent sets into a single set. Thus the size of the partition decrements by at least one; hence, the lemma follows. ■

Now we have the necessary ingredients to prove termination below.

Theorem 1. *The Compositional Learning Algorithm terminates.*

Proof. Assume, towards contradiction, that the algorithm does not terminate. Let I be the alphabet, an I_k^F be the partition of I after the k^{th} round of the algorithm. By Lemma 4, after at least $k = |I| - 1$ rounds, $|I_k^F| = 1$. Also by the assumption, the algorithm has not terminated at round k. Since $I_k^F = I$, the algorithm reduces to algorithm L^* which terminates. Hence, the contradiction. ■

We prove next that every time we merge two partitions, there is a sound reason (i.e., dependency of actions) for it.

Theorem 2. *Let* `CE` *be the minimal counter-example returned by the oracle at round k of the algorithm and $I^F = \{I_1, \ldots, I_n\}$ the partition of the alphabet at the same round. Then, all actions in* `CE` *are dependent.*

Proof. Let $\texttt{CE} = wa$, $w \in I^*$ and $a \in I$, and $d = \{d_1, \ldots, d_m\}$ be an independent partition for the SUL M. Assume some actions in w are independent from a (proof by contradiction). Let d_k be the set in d that includes a. The set $I \setminus d_k$ contains all the independent actions from a. For M, we define $O_M = P_{d_k}(\lambda_M(wa))$; according to Lemma 2, $O_M = \lambda_{P(M,d_k)}(P_{d_k}(wa))$. The algorithm makes the hypothesis $\mathscr{H} = P(M, I_1)||\ldots||P(M, I_n)$ at the current round k. Since d_k is the union of a subset of I^F (algorithm has not terminated yet), $O_{\mathscr{H}} = P_{d_k}(\lambda_{\mathscr{H}}(wa)) = \lambda_{P(\mathscr{H},d_k)}(P_{d_k}(wa))$. If $O_{\mathscr{H}} \neq O_M$, then $P_{d_k}(wa)$ is a smaller counter-example than wa, which is a contradiction. Otherwise if $O_{\mathscr{H}} = O_M$, given that wa is a counter-example, $P_{I\setminus d_k}(\lambda_M(wa)) \neq P_{I\setminus d_k}(\lambda_{\mathscr{H}}(wa))$; if so, $P_{I\setminus d_k}(wa)$ is a smaller counter-example, hence the contradiction. ■

By Theorems 2 and 1, we have shown that the algorithm detects the independent action sets and eventually terminates. The next theorem is formulated to show that it terminates as soon as all dependent action sets have been detected.

Theorem 3. *Let $I^F = \{I_1, \ldots, I_n\}$ be an independent partition of the alphabet at round k. The algorithm terminates in this round.*

Proof. We prove this by contradiction. Assume that the algorithm does not terminate, and CE is the minimal counter-example returned by the oracle. By theorem 2, *InvolvedSets* returns two or more dependent sets from I^F. Since all the elements in I^F are pairwise independent, we confront the contradiction. ■

4.2 Processing Counter-examples

As mentioned in Theorem 2, we require all the actions in a minimal counter-example returned by the oracle to be dependent. However, most equivalence checking methods do not find the minimal counter-example. For a non-minimal counter-example, we define a process called "distillation", which asks a number of extra queries to find the dependent actions. It iteratively gets a subset of *InvolvedSets*(CE, I^F) in the order of their sizes and merges its members together, producing a set M. The algorithm introduces $P_M(\texttt{CE})$ as output if it is a counter-example.

Suppose CE is the counter-example returned by the oracle at round k of the algorithm, and I^F is the alphabet partition at that round. To distill two or more dependent sets from CE, we follow Algorithm 2. The function *CutCE* on line 2 takes a counter-example CE and returns the smallest prefix of CE, which is also a counter-example (i.e., the SUL and the hypothesis model produce different outputs for it). Then, iteratively, it gets a subset of *InvolvedSets*(CE, I^F) in the order of their sizes and merges its members together, producing set M. The algorithm returns $P_M(\texttt{CE})$ as output if it is a counter-example.

The cost of CE-distillation algorithms is exponential in terms of the size of CE in the worst case. However, in the results section, we show that in practice, the cost of this part is not very significant compared to the total cost of learning.

Theorem 4. *All actions in the output of the CE distillation algorithm are dependent.*

The proof is omitted as it is similar to the proof of Theorem 2.

Algorithm 2: CE distillation

Result: $\texttt{CE}_M$
Input: $I^F = \{I_1, \ldots, I_n\}$, CE, M, $\mathscr{H}$
CE $\leftarrow CutCE(\texttt{CE})$
$D \leftarrow InvolvedSets(\texttt{CE}, I^F)$
for $k \in \{2, \ldots, size(D)\}$ **do**
  $C \leftarrow$ all k combinations(D)
  while C is not empty **do**
    $I \leftarrow C.pop$
    $A \leftarrow \bigcup_{i \in I} I_i$
    $\texttt{CE}_A \leftarrow P_A(\texttt{CE})$
    if $\texttt{CE}_A$ is a counter-example **then**
      Return $\texttt{CE}_A$
    end
  end
end

5 Empirical Evaluation

In this section, we present the design and the results of the experiments carried out to evaluate our approach, in order to answer the following research questions:

RQ1 Does CL* require fewer resets, compared to L*?
RQ2 Does CL* require fewer input symbols, compared to L*?

As stated in Section 1, these two research questions measure the efficiency of a learning method in a machine-independent manner: the number of input symbols summarises the total cost of a learning campaign, while the number of resets summarises one of its most costly parts. Note that although active learning processes are structured in terms of queries, the queries used in the processes have vastly different lengths and it has been observed earlier that the total number of input symbols is a more accurate metric for comparison of learning algorithms than the number queries [36].

5.1 Subject Systems

A meaningful benchmark for our method should feature systems of various state sizes and various numbers of parallel components and with a non-trivial structure that may require multiple learning rounds. Also, we would like to have realistic systems, so that our comparisons have meaningful practical implications.

To this end, we choose the Body Comfort System (BCS) [25], which is an automotive software product line (SPL) of a Volkswagen Golf model. This SPL has 27 components, each representing a feature that provides specific functionality. The transition system of each component is provided in a detailed technical report [24]. We use the finite state machines of the components constructed from

the transition system representations in [35] and compose several random samples utilising the interleaving parallel composition (Definition 2) to build the product FSMs. We automatically constructed 100 FSMs consisting of a minimum of two and a maximum of nine components in this case study. The maximum number is chosen due the performance limits of L*; beyond this limit, our learning campaign for L* could take more than four hours. All experiments were conducted on a computer with an Intel® Core™ M-5Y10c CPU and and 8GB of physical memory running Ubuntu version 20 and LearnLib version 0.16.0. Our subject systems have a minimum of 300 states and a maximum of 3840 states, and their average number of states is 1278.2 with a standard deviation of 847. We started the calculation of the metrics for subject systems of at least 300 states, since for small subject systems, the advantage of compositional learning is not significant.

5.2 Experiment Design

To answer the research questions, we implemented the compositional learning algorithm on top of the LearnLib framework [30]. This implementation uses the equivalence oracle in two places; to learn projections in the *LearnInParts* function and to check the hypothesis/SUL equivalence. The performance of the algorithm significantly relies on the type of equivalence queries used by the underlying L* algorithm. We experimented with a number of equivalence methods and settled upon using random walks; when using deterministic algorithms such as the WP- and the WP-method, for large systems, the cost of equivalence queries becomes prohibitively high and obscures any gain obtained from compositionality. To ensure that our results are sound, we have carried out similar experiments by using an additional deterministic equivalence query at the end of the learning campaign, when the last random equivalence query does not return any counter-example. This additional step verifies our comparisons when an assurance about the accuracy of the learning process is required. More details about these additional experiments can be found in our public lab package [23] (`https://github.com/faezeh-lbf/CL-Star`).

We enabled caching, since caching significantly reduces repetitive queries. We repeat each learning process three times, comparing the number of resets and input symbols for L* and CL*.

In addition to reporting the median metrics, their standard deviations, and the relative percentage of improvements, we use the statistical T-test to answer the research questions with statistical confidence and report the p-values. We analyse the distribution of the results and establish their normality using K-tests. We use the SciPy [20] library of Python to perform statistical analysis and Seaborn [38] for visualising the results.

5.3 Results

In this section, we first present the results of our experiments and use them to answer our research questions. Then we show how the number of components in

an FSM affects the efficiency of our algorithm. Finally, we discuss threats to the validity of our empirical results.

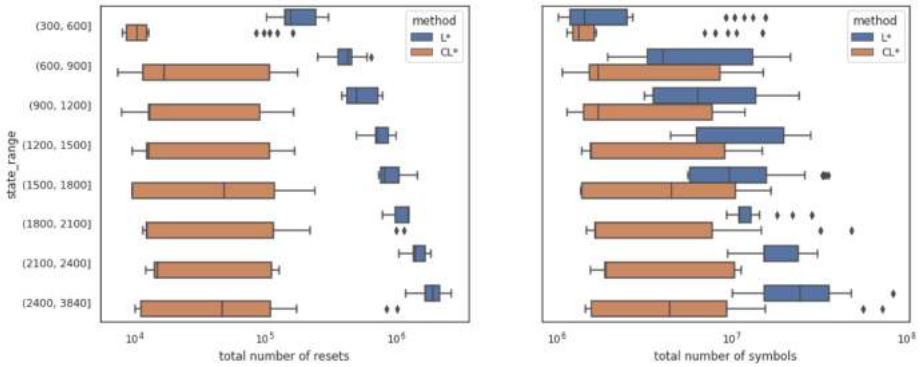


Fig. 2: The total number of input symbols and resets in the CL* and L* methods

We cluster the benchmark into eight categories based on the FSM's number of states and illustrate the distribution of input symbols and resets for each cluster in Figure 2. In this figure, the CL* and L* methods are compared based on the metrics mentioned. The scale of the x-axis (the value of metrics) is logarithmic.

Tables 1 and 2 summarise the results of our experiments. For each category, we calculate the median and standard deviation of our metrics (the number of input symbols and resets) both for L* and CL*. The metric "progress percentage" is defined to measure the improvement brought about by compositional learning (compared to L*). For each metric, the progress percentage is calculated as $(1 - \frac{p}{q}) * 100$, where p and q are the value of that metric in CL* and L*, respectively. A positive progress percentage in a metric shows that the CL* is more efficient in terms of that metric. To measure the statistical significance, we used the one-sided paired sample T-test to check if there was a significant difference ($p < 0.05$) between the metrics in the two algorithms.

Table 1: Comparing the total number of input symbols in the CL* and L* methods

#States	L* method		CL* method		Progress	p-value
	Median	Standard deviation	Median	Standard deviation	percentage	(one-sided paired T-test)
(300, 600]	1443710	2834380.581	1329818	2382620.467	14.47	7.43e-3
(600, 900]	4013396	6262292.443	1716878.5	4408369.926	36.44	1.54e-8
(900, 1200]	6387472	6663334.645	1714934.5	3757307.024	52.37	8.36e-7
(1200, 1500]	6259466	9311767.302	1576494	4798094.639	57.28	6.49e-4
(1500, 1800]	9700935	10726103.24	4498072	5576873.639	54.58	4.30e-4
(1800, 2100]	11070428	5310108.013	1649557	13958718.62	37.51	2.96e-2
(2100, 2400]	15348181	6287714.182	1888226	4215184.514	70.80	1.80e-10
(2400, 3840]	24700222.5	14837416.08	4385086	13817389.06	68.42	2.66e-12

Table 2: Comparing the total number of resets in the CL* and L* methods

#States	L* method		CL* method		Progress	p-value
	Median	Standard deviation	Median	Standard deviation	percentage	(one-sided paired T-test)
(300, 600]	157971	65257.85738	10433	28259.60196	90.46	1.05e-33
(600, 900]	425260.5	77944.01883	16808	56274.51558	86.33	1.07e-43
(900, 1200]	501347.5	147915.8363	13109	50224.87222	90.87	3.80e-16
(1200, 1500]	712999	136904.04	12811	60125.8884	91.77	4.18e-13
(1500, 1800]	823482	275862.8299	48344	80507.59837	91.73	4.97e-13
(1800, 2100]	1262025	188390.1181	12412	369932.964	84.07	2.18e-06
(2100, 2400]	1412237	220211.8459	15042	53006.08784	95.83	2.44e-14
(2400, 3840]	1900234	427883.9888	46624.5	201052.8807	94.67	2.20e-23

Both Tables 1 and 2 indicate major improvements, particularly for large systems, in terms of the total number of input symbols and resets, respectively. Compositional learning reduces the number of symbols up to 70.80 percent and the number of resets up to 95.83 percent. The statistical tests also confirm this observations and the p-values obtained from the tests are in all cases very low; in case of the number of input symbols the p-values range from 10^{-2} to 10^{-12}, while for resets they range from 10^{-6} to 10^{-43}, which are well-below the usual statistical p-values (0.05) and represent a very high statistical significance.

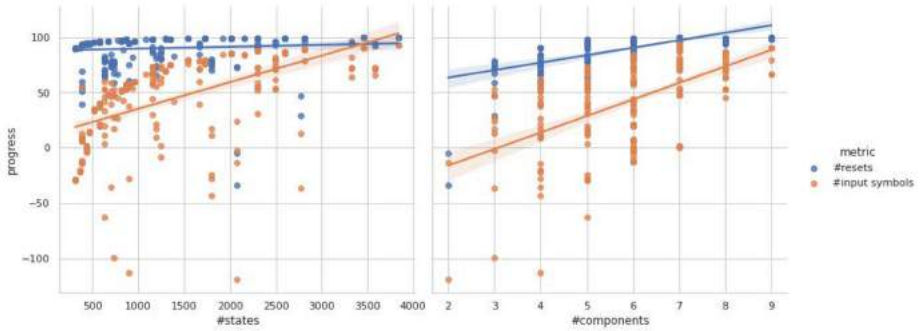


Fig. 3: The diagrams of improvement brought about by compositional learning vs. size of the SUL in terms of number states (left) and components (right).

The plots in Figure 3 visualise the improvements brought about by compositional learning. This plot demonstrates that the saving due to compositional learning increases as the number of components in SULs increases. We further analysed the trends of our measured metrics in terms of the number of states and the number of parallel components. These trends are depicted for the total number of input symbols in Figure 4 and for the number of resets in Figure 5, respectively. These figures indicate that the increase of both metrics with the number of states is more moderate for the compositional learning approach, i.e., compositional learning is more scalable. More importantly, the right-hand-side

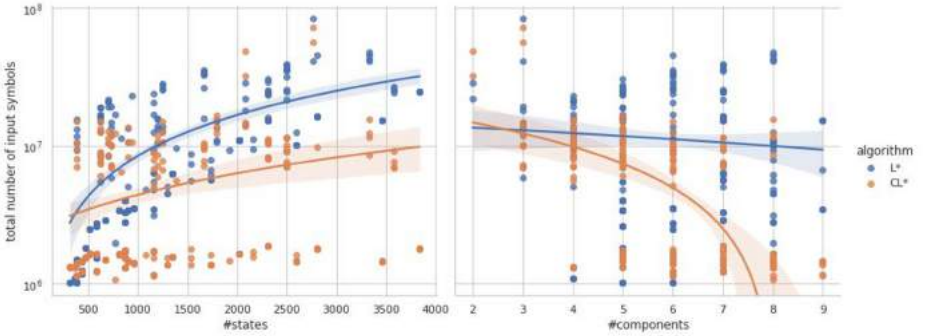


Fig. 4: The effect of FSM sizes in terms of the number of components and states on the total number of input symbols.

of both figures signifies the effect of compositional learning when the number of parallel components increases while the number of states remains fixed.

Figure 6 shows the effect of the number of components on the total number of input symbols for a fixed state-space size for algorithms L* and CL*. In this plot, as the number of components increases, the corresponding dot will become darker and larger. According to this figure, the learning cost is lower for SULs with more components in both L* and CL*. Still, for CL* (the right side), the cost of learning SULs with more components is significantly lower because we structurally learn these components essentially independently.

As mentioned in Section 4.2, the cost of the CE distillation process can increase exponentially in the size of the counter-example. However, in practice, it seems to be much more tractable. To evaluate this, we count the number of input symbols required by the CE distillation process to learn each SUL. The median value of this metric is 1961 input symbols, which is insignificant compared the total cost of learning. In fact, the cost of CE distillation process for each group in Table 1 is between 0.037 and 0.12 percent of the total learning cost; the reported total learning cost (total number of input symbols) includes the cost of CE distillation.

5.4 Threats to Validity

In this section, we summarise the major threats to the validity of our empirical conclusions. First, we analyse the threats to conclusion validity, i.e., whether the empirical conclusions necessarily follow from the experiments carried out. Then, we discuss the threats to external validity concerning the generalisation of our results to other systems.

We mitigated conclusion validity threats by using statistical tests to ensure that our observations (both in terms of improvement percentages in Tables 1 and 2 and the visual observations in Figures 2) do represent a statistically significant improvement. We opt for one-sided paired sample T-tests in order to minimise

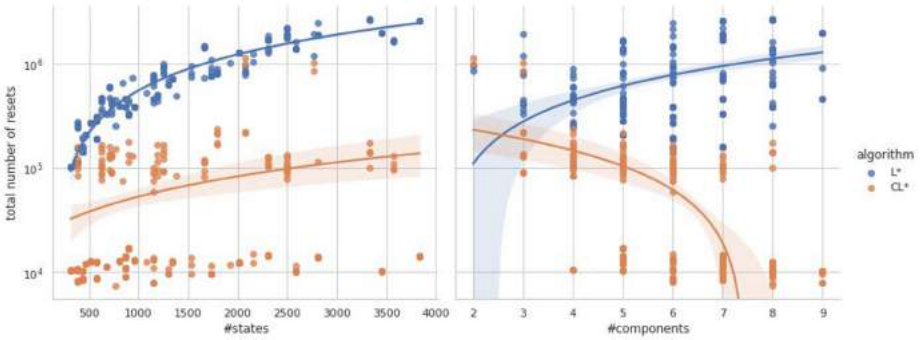


Fig. 5: The effect of the size of FSMs in terms of the number of components and states on the total number of required input resets.

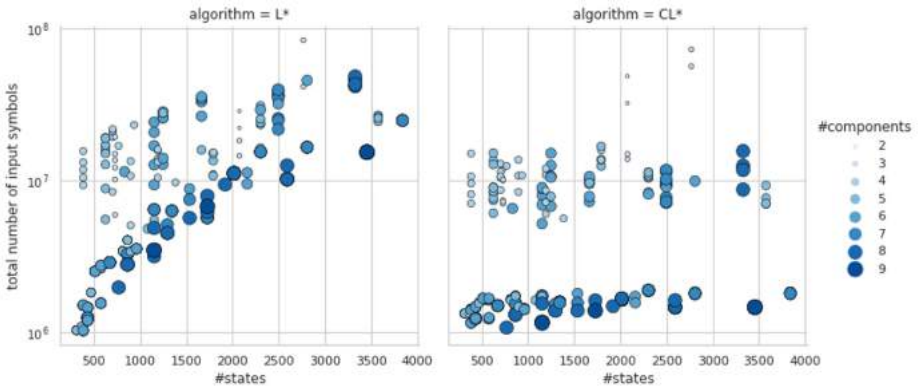


Fig. 6: The relation between the total number of symbols and the number of states and components for the algorithms L* (left) and CL* (right).

the threats to conclusion validity. We only conclude that the CL* is more efficient than the L* when there is a meaningful difference ($p < 0.05$) between the results of L* and CL*. To make sure that the chosen statistical test is applicable, we analysed the distribution of the data first.

We mitigated the risk of conclusion validity by using subject systems that are based on practical systems rather than using randomly generated FSMs. However, further research is needed to analyse the performance of our approach based on other benchmarks from other domains. We also mitigated the effect of using random equivalence queries by repeating the experiments with a final deterministic query.

6 Conclusions

In this paper, we presented a compositional learning method based on Angluin's algorithm L^* that detects and independently learns interleaving parallel components of the system under learning. We proved that our algorithm, called CL^*, is correct and we empirically showed that it causes significant gains in the number of input symbols and the number of resets in a learning campaign. The gain is significantly increased with the number of parallel components.

Our algorithm is naturally amenable to parallelisation and developing a parallel implementation is a natural next step. A more thorough investigation of counter-example processing in order to efficiently find a minimal counter-example is an area of further research, particularly, in the light of the recent results in this area [13]. Finding a trade-off between using deterministic and random (or mutation-based) equivalence queries is another area of future research. We would also like to investigate the possibility of developing equivalence queries that take the structure of the systems into account: we have observed that much of the effort in the final equivalence query (on the composed system) is redundant and the final equivalence query can be made much more efficient by only considering the dependencies among purportedly independent partitions. Finally, extending our notion of parallel composition to allow for a possible synchronisation of components is another direction of future work; we believe inspirations from concurrency theory and in particular, Milner and Moller's prime decomposition theorem [26] may prove effective in this regard. Independently from our work, Neele and Sammartino [29] proposed an approach to learn synchronous parallel composition, under the assumption of knowing the alphabets of the components. This is a promising approach to incorporate synchronous parallel composition into our framework.

Acknowledgments

We would like to thank Rasta Tadayon and Amin Asadi Sarijalou for their contributions to the early stages of this work. The work of Mohammad Reza Mousavi was supported by the UKRI Trustworthy Autonomous Systems Node in Verifiability, Grant Award Reference EP/V026801/2. We thank the reviewers of FOSSACS for their insightful and constructive comments, which, in our view, led to improvements in our final paper. We thank the Artifact Evaluation committee at ESOP/FOSSACS for their careful review of our lab package.

References

1. Aarts, F., de Ruiter, J., Poll, E.: Formal models of bank cards for free. In: Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013 Workshops Proceedings, Luxembourg, Luxembourg, March 18-22, 2013. pp. 461–468. IEEE Computer Society (2013). https://doi.org/10.1109/ICSTW.2013.60

2. Aarts, F., Schmaltz, J., Vaandrager, F.W.: Inference and abstraction of the biometric passport. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification, and Validation - 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part I. Lecture Notes in Computer Science, vol. 6415, pp. 673–686. Springer (2010). https://doi.org/10.1007/978-3-642-16558-0_54
3. Aichernig, B.K., Tappler, M.: Efficient active automata learning via mutation testing. Journal of Automated Reasoning **63**(4), 1103–1134 (2019). https://doi.org/10.1007/s10817-018-9486-0
4. Aichernig, B.K., Tappler, M., Wallner, F.: Benchmarking combinations of learning and testing algorithms for active automata learning. In: Ahrendt, W., Wehrheim, H. (eds.) Tests and Proofs - 14th International Conference, TAP@STAF 2020, Bergen, Norway, June 22-23, 2020, Proceedings [postponed]. Lecture Notes in Computer Science, vol. 12165, pp. 3–22. Springer (2020). https://doi.org/10.1007/978-3-030-50995-8_1
5. An, J., Chen, M., Zhan, B., Zhan, N., Zhang, M.: Learning one-clock timed automata. In: Biere, A., Parker, D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 444–462. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-45190-5_25
6. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6
7. Broy, M., Jonsson, B., Katoen, J., Leucker, M., Pretschner, A. (eds.): Model-Based Testing of Reactive Systems, Advanced Lectures [The volume is the outcome of a research seminar that was held in Schloss Dagstuhl in January 2004], Lecture Notes in Computer Science, vol. 3472. Springer (2005). https://doi.org/10.1007/b137241
8. Cifuentes, C., Simon, D.: Procedure abstraction recovery from binary code. In: Proceedings of the Fourth European Conference on Software Maintenance and Reengineering. pp. 55–64. IEEE (2000). https://doi.org/10.1109/CSMR.2000.827306
9. Damasceno, C.D.N., Mousavi, M.R., da Silva Simão, A.: Learning by sampling: learning behavioral family models from software product lines. Empir. Softw. Eng. **26**(1), 4 (2021). https://doi.org/10.1007/s10664-020-09912-w
10. al Duhaiby, O., Groote, J.F.: Active learning of decomposable systems. In: Bae, K., Bianculli, D., Gnesi, S., Plat, N. (eds.) FormaliSE@ICSE 2020: 8th International Conference on Formal Methods in Software Engineering, Seoul, Republic of Korea, July 13, 2020. pp. 1–10. ACM (2020). https://doi.org/10.1145/3372020.3391560
11. Fiterau-Brostean, P., Lenaerts, T., Poll, E., de Ruiter, J., Vaandrager, F.W., Verleg, P.: Model learning and model checking of SSH implementations. In: Erdogmus, H., Havelund, K. (eds.) Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017. pp. 142–151. ACM (2017). https://doi.org/10.1145/3092282.3092289
12. Frohme, M., Steffen, B.: Compositional learning of mutually recursive procedural systems. Int. J. Softw. Tools Technol. Transf. **23**(4), 521–543 (2021). https://doi.org/10.1007/s10009-021-00634-y
13. Frohme, M., Steffen, B.: From languages to behaviors and back. In: Jansen, N., Stoelinga, M., van den Bos, P. (eds.) A Journey from Process Algebra via Timed Automata to Model Learning - Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 13560, pp. 180–200. Springer (2022). https://doi.org/10.1007/978-3-031-15629-8_11
14. Garhewal, B., Vaandrager, F.W., Howar, F., Schrijvers, T., Lenaerts, T., Smits, R.: Grey-box learning of register automata. In: Dongol, B., Troubitsyna, E. (eds.)

Integrated Formal Methods - 16th International Conference, IFM 2020, Lugano, Switzerland, November 16-20, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12546, pp. 22–40. Springer (2020). https://doi.org/10.1007/978-3-030-63461-2_2

15. Hooimeijer, B., Geilen, M., Groote, J.F., Hendriks, D., Schiffelers, R.R.H.: Constructive model inference: Model learning for component-based software architectures. In: Fill, H., van Sinderen, M., Maciaszek, L.A. (eds.) Proceedings of the 17th International Conference on Software Technologies, ICSOFT 2022, Lisbon, Portugal, July 11-13, 2022. pp. 146–158. SCITEPRESS (2022). https://doi.org/10.5220/0011145700003266
16. Howar, F., Steffen, B.: Active automata learning in practice - an annotated bibliography of the years 2011 to 2016. In: Bennaceur, A., Hähnle, R., Meinke, K. (eds.) Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers. Lecture Notes in Computer Science, vol. 11026, pp. 123–148. Springer (2018). https://doi.org/10.1007/978-3-319-96562-8_5
17. Howar, F., Steffen, B.: Active automata learning as black-box search and lazy partition refinement. In: Jansen, N., Stoelinga, M., van den Bos, P. (eds.) A Journey from Process Algebra via Timed Automata to Model Learning : Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday, pp. 321–338. Springer Nature Switzerland, Cham (2022). https://doi.org/10.1007/978-3-031-15629-8_17
18. Isberner, M., Howar, F., Steffen, B.: Learning register automata: from languages to program structures. Machine Learning **96**(1), 65–98 (2014). https://doi.org/10.1007/s10994-013-5419-7
19. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: A redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8734, pp. 307–322. Springer (2014). https://doi.org/10.1007/978-3-319-11164-3_26
20. Jones, E., Oliphant, T., Peterson, P.: Scipy: Open source scientific tools for python (01 2001). https://doi.org/10.1038/s41592-019-0686-2
21. Kearns, M.J., Vazirani, U.: An Introduction to Computational Learning Theory. MIT Press (1994). https://doi.org/10.7551/mitpress/3897.001.0001
22. Koschke, R.: Architecture Reconstruction, p. 140–173. Springer-Verlag, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-540-95888-8_6
23. Labbaf, F., Groot, J.F., Hojjat, H., Mousavi, M.R.: Compositional Learning for Interleaving Parallel Automata (CL-Star) (Apr 2023). https://doi.org/10.5281/zenodo.7624699, https://doi.org/10.5281/zenodo.7624699
24. Lachmann, R., Lity, S., Lischke, S., Beddig, S., Schulze, S., Schaefer, I.: Delta-oriented test case prioritization for integration testing of software product lines. In: Proceedings of the 19th International Conference on Software Product Line. p. 81–90. SPLC '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2791060.2791073
25. Lity, S., Lachmann, R., Lochau, M., Schaefer, I.: Delta-oriented software product line test models-the body comfort system case study. Tech. Rep. 2012-07, TU Braunschweig (2012)
26. Milner, R., Moller, F.: Unique decomposition of processes. Theoretical Computer Science **107**(2), 357–363 (1993). https://doi.org/10.1016/0304-3975(93)90176-T, https://www.sciencedirect.com/science/article/pii/030439759390176T

27. Moerman, J.: Learning product automata. In: Unold, O., Dyrka, W., Wieczorek, W. (eds.) Proceedings of The 14th International Conference on Grammatical Inference 2018. Proceedings of Machine Learning Research, vol. 93, pp. 54–66. PMLR (feb 2019), https://proceedings.mlr.press/v93/moerman19a.html
28. Naeem Irfan, M., Oriat, C., Groz, R.: Model inference and testing. Advances in Computers, vol. 89, pp. 89–139. Elsevier (2013). https://doi.org/10.1016/B978-0-12-408094-2.00003-5, https://www.sciencedirect.com/science/article/pii/B9780124080942000035
29. Neele, T., Sammartino, M.: Compositional Automata Learning of Synchronous Systems. In: Lambers, L., Uchitel, S. (eds.) FASE 2023. Lecture Notes in Computer Science, Springer (2023)
30. Raffelt, H., Steffen, B.: Learnlib: A library for automata learning and experimentation. In: Baresi, L., Heckel, R. (eds.) Fundamental Approaches to Software Engineering. pp. 377–380. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). https://doi.org/10.1145/1081180.1081189
31. Rivest, R., Schapire, R.: Inference of finite automata using homing sequences. Information and Computation **103**(2), 299–347 (1993). https://doi.org/10.1006/inco.1993.1021
32. Sanchez, L., Groote, J.F., Schiffelers, R.R.H.: Active learning of industrial software with data. In: Hojjat, H., Massink, M. (eds.) Fundamentals of Software Engineering - 8th International Conference, FSEN 2019, Tehran, Iran, May 1-3, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11761, pp. 95–110. Springer (2019). https://doi.org/10.1007/978-3-030-31517-7_7
33. Smeenk, W., Moerman, J., Vaandrager, F.W., Jansen, D.N.: Applying automata learning to embedded control software. In: Butler, M.J., Conchon, S., Zaïdi, F. (eds.) Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015, Paris, France, November 3-5, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9407, pp. 67–83. Springer (2015). https://doi.org/10.1007/978-3-319-25423-4_5
34. Tappler, M., Aichernig, B.K., Larsen, K.G., Lorber, F.: Time to learn – learning timed automata from tests. In: Formal Modeling and Analysis of Timed Systems: 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27–29, 2019, Proceedings. p. 216–235. Springer-Verlag, Berlin, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29662-9_13
35. Tavassoli, S., Damasceno, C.D.N., Khosravi, R., Mousavi, M.R.: Adaptive behavioral model learning for software product lines. In: Felfernig, A., Fuentes, L., Cleland-Huang, J., Assunção, W.K.G., Falkner, A.A., Azanza, M., Luaces, M.Á.R., Bhushan, M., Semini, L., Devroey, X., Werner, C.M.L., Seidl, C., Le, V., Horcas, J.M. (eds.) SPLC '22: 26th ACM International Systems and Software Product Line Conference, Graz, Austria, September 12 - 16, 2022, Volume A. pp. 142–153. ACM (2022). https://doi.org/10.1145/3546932.3546991
36. Vaandrager, F.: Model learning. Commun. ACM **60**(2), 86–95 (jan 2017). https://doi.org/10.1145/2967606
37. Vaandrager, F.W., Garhewal, B., Rot, J., Wißmann, T.: A new approach for active automata learning based on apartness. In: Fisman, D., Rosu, G. (eds.) Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS 2022. Lecture Notes in Computer Science, vol. 13243, pp. 223–243. Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_12
38. Waskom, M.L.: seaborn: statistical data visualization. Journal of Open Source Software **6**(60), 3021 (2021). https://doi.org/10.21105/joss.03021

Pebble minimization: the last theorems

Gaëtan Douéneau-Tabot(✉)

[1] Université Paris Cité, CNRS, IRIF, F-75013, Paris, France
[2] Direction générale de l'armement - Ingénierie des projets, Paris, France
douenau@irif.fr

Abstract Pebble transducers are nested two-way transducers which can drop marks (named "pebbles") on their input word. Such machines can compute functions whose output size is polynomial in the size of their input. They can be seen as simple recursive programs whose recursion height is bounded. A natural problem is, given a pebble transducer, to compute an equivalent pebble transducer with minimal recursion height. This problem has been open since the introduction of the model.
In this paper, we study two restrictions of pebble transducers, that cannot see the marks ("blind pebble transducers" introduced by Nguyên et al.), or that can only see the last mark dropped ("last pebble transducers" introduced by Engelfriet et al.). For both models, we provide an effective algorithm for minimizing the recursion height. The key property used in both cases is that a function whose output size is linear (resp. quadratic, cubic, etc.) can always be computed by a machine whose recursion height is 1 (resp. 2, 3, etc.). We finally show that this key property fails as soon as we consider machines that can see more than one mark.

Keywords: Pebble transducers · Polyregular functions · Blind pebble transducers · Last pebble transducers · Factorization forests.

1 Introduction

Transducers are finite-state machines obtained by adding outputs to finite automata. They are very useful in a lot of areas like coding, computer arithmetic, language processing or program analysis, and more generally in data stream processing. In this paper, we consider deterministic transducers which compute functions from finite words to finite words. In particular, a **deterministic two-way transducer** is a two-way automaton with outputs. This model describes the class of **regular functions**, which is often considered as one of the functional counterparts of regular languages. It has been intensively studied for its properties such as closure under composition [5], equivalence with logical transductions [12] or regular expressions [7], decidable equivalence problem [14], etc.

Pebble transducers and polyregular functions. Two-way transducers can only describe functions whose output size is at most linear in the input size. A possible solution to overcome this limitation is to consider nested two-way

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 436–455, 2023.
https://doi.org/10.1007/978-3-031-30829-1_21

transducers. In particular, the model of **k-pebble transducer** has been studied for a long time [13]. For $k = 1$, a 1-pebble transducer is just a two-way transducer. For $k \geqslant 2$, a k-pebble transducer is a two-way transducer that, when on any position i of its input word, can call a $(k-1)$-pebble transducer. The latter takes as input the original input where position i is marked by a "pebble". The main two-way transducer then outputs the concatenation of all the outputs produced along its calls. The intuitive behavior of a 3-pebble transducer is depicted in fig. 1. It can be seen as a recursive program whose recursion stack has height 3. The class of functions computed by pebble transducers is known as **polyregular functions**. It has been intensively studied due to its properties such as closure under composition [11], equivalence with logical interpretations [4], etc.

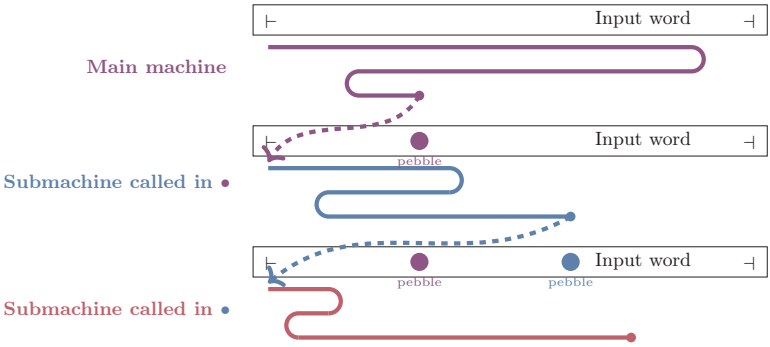


Figure 1: Behavior of a 3-pebble transducer.

Optimization of pebble transducers. Given a k-pebble transducer computing a function f, a very natural problem is to compute the least possible $1 \leqslant \ell \leqslant k$ such that f can be computed by an ℓ-pebble transducer. Furthermore, we can be interested in effectively building an ℓ-pebble transducer for f. Both questions are open, but they are meaningful since they ask whether we can optimize the recursion height (i.e. the running time) of a program.

It is easy to observe that if f is computed by a k-pebble transducer, then $|f(u)| = \mathcal{O}(|u|^k)$. It was first claimed in a LICS 2020 paper that the minimal recursion height ℓ of f (i.e. the least possible ℓ such that f can be computed by an ℓ-pebble transducer) was exactly the least possible ℓ such that $|f(u)| = \mathcal{O}(|u|^\ell)$. However, Bojańczyk recently disproved this statement in [3, Theorem 6.3]: the function $\mathsf{inner\text{-}squaring} : u_1\# \cdots \#u_n \mapsto (u_1\#)^n \cdots (u_n\#)^n$ can be computed by a 3-pebble transducer and is such that $|\mathsf{inner\text{-}squaring}(u)| = \mathcal{O}(|u|^2)$, but it cannot be computed by a 2-pebble transducer. Other counterexamples were given in [16] using different proof techniques. Therefore, computing the minimal recursion height of f is believed to be hard, since this value not only depends on the output size of f, but also on the word combinatorics of this output.

Optimization of blind pebble transducers. A subclass of pebble transducers, named **blind pebble transducers**, was recently introduced in [17]. A blind k-pebble transducer is somehow a k-pebble transducer, with the difference that the positions are no longer marked when making recursive calls. The behavior of a blind 3-pebble transducer is depicted in fig. 2. The class of functions computed by blind pebble transducers is strictly included in polyregular functions [10,17]. The main result of [17] shows that for blind pebble transducers, the minimal recursion height for computing a function only depends on the growth of its output. More precisely, if f is computed by a blind k-pebble transducer, then the least possible $1 \leqslant \ell \leqslant k$ such that f can be computed by an blind ℓ-pebble transducer is the least possible ℓ such that $|f(u)| = \mathcal{O}(|u|^\ell)$.

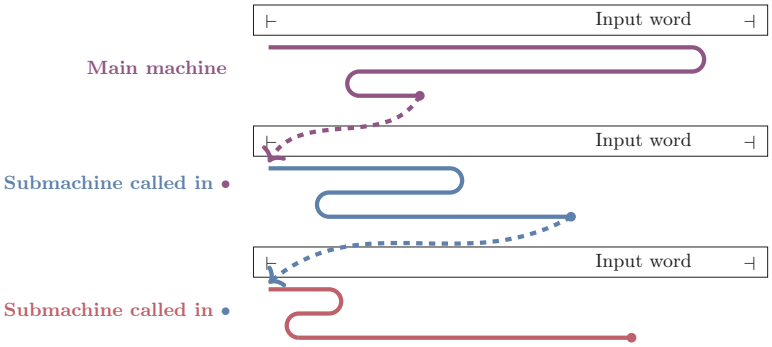


Figure 2: Behavior of a blind 3-pebble transducer.

Contributions. In this paper, we first give a new proof of the connection between minimal recursion height and growth of the output for blind pebble transducers. Furthermore, our proof provides an algorithm that, given a function computed by a blind k-pebble transducer, builds a blind ℓ-pebble transducer which computes it, for the least possible $1 \leqslant \ell \leqslant k$. This effective result is not claimed in [17], and our proof techniques significantly differ from theirs. Indeed, we make a heavy use of **factorization forests**, which have already been used as a powerful tool in the study of pebble transducers [2,8,10].

Secondly, the main contribution of this paper is to show that the (effective) connection between minimal recursion height and growth of the output also holds for the class of **last pebble transducers** (introduced in [13]). Intuitively, a last k-pebble transducer is a k-pebble transducer where a called submachine can only see the position of its call, but not the full stack of the former positions. The behavior of a last 3-pebble transducer is depicted in fig. 3. Observe that a blind k-pebble transducer is a restricted version of a last k-pebble transducer. Formally, we show that if f is computed by a last k-pebble transducer, then the least possible ℓ such that f can be computed by a last ℓ-pebble transducer is the least possible ℓ such that $|f(u)| = \mathcal{O}(|u|^\ell)$. Furthermore, our proof gives an algorithm that effectively builds a last ℓ-pebble transducer computing f.

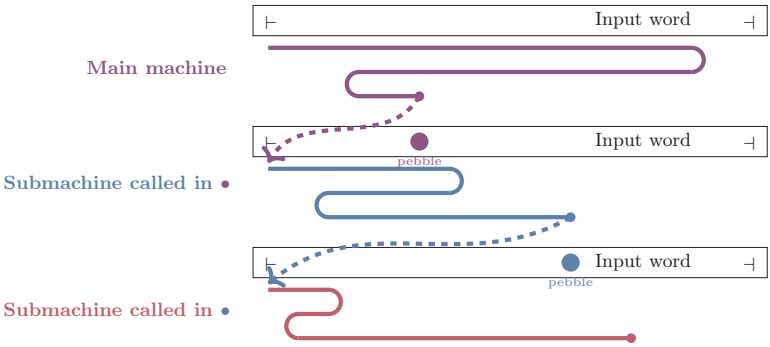


Figure 3: Behavior of a last 3-pebble transducer.

As a third theorem, we show that our result for last pebble transducers is tight, in the sense that the connection between minimal recursion height and growth of the output does not hold for more powerful models. More precisely, we define the model of **last-last k-pebble transducers**, which extends last k-pebble transducers by allowing them to see the two last positions of the calls (and not only the last one). We show that for all $k \geqslant 1$, there exists a function f such that $|f(u)| = \mathcal{O}(|u|^2)$ and that is computed by a last-last $(2k{+}1)$-pebble transducer, but cannot be computed by a last-last $2k$-pebble transducer. The proof of this result relies on a counterexample presented by Bojańczyk in [2].

Outline. We introduce two-way transducers in section 2. In section 3 we describe blind pebble transducers and last pebble transducers. We also state our main results that connect the minimal recursion height of a function to the growth of its output. Their proof goes over sections 4 to 6. In section 7, we finally show that these results cannot be extended to two visible marks.

2 Preliminaries on two-way transducers

Capital letters A, B denote alphabets, i.e. finite sets of letters. The empty word is denoted by ε. If $u \in A^*$, let $|u| \in \mathbb{N}$ be its length, and for $1 \leqslant i \leqslant |u|$ let $u[i]$ be its i-th letter. If $i \leqslant j$, we let $u[i{:}j]$ be $u[i]u[i{+}1]\cdots u[j]$ (empty if $j < i$). If $a \in A$, let $|u|_a$ be the number of letters a occurring in u. We assume that the reader is familiar with the basics of automata theory, in particular two-way automata and monoid morphisms. The type of total (resp. partial, i.e. possibly undefined on some inputs) functions is denoted $S \to T$ (resp. $S \rightharpoonup T$).

The machines described in this paper are always **deterministic**.

Definition 2.1. *A **two-way transducer** $\mathscr{T} = (A, B, Q, q_0, F, \delta, \lambda)$ consists of:*
- *an input alphabet A and an output alphabet B;*
- *a finite set of states Q with $q_0 \in Q$ initial and $F \subseteq Q$ final;*
- *a transition function $\delta : Q \times (A \uplus \{\vdash, \dashv\}) \rightharpoonup Q \times \{\triangleleft, \triangleright\}$;*
- *an output function $\lambda : Q \times (A \uplus \{\vdash, \dashv\}) \rightharpoonup B^*$ with same domain as δ.*

The semantics of a two-way transducer $\mathscr{T}$ is defined as follows. When given as input a word $u \in A^*$, $\mathscr{T}$ disposes of a read-only input tape containing $\vdash u \dashv$. The marks $\vdash$ and $\dashv$ are used to detect the borders of the tape, by convention we denote them by positions 0 and $|u|{+}1$ of u. Formally, a configuration over $\vdash u \dashv$ is a tuple (q, i) where $q \in Q$ is the current state and $0 \leqslant i \leqslant |u|{+}1$ is the position of the reading head. The transition relation $\to$ is defined as follows. Given a configuration (q, i), let $(q', \star) := \delta(q, u[i])$. Then $(q, i) \to (q', i')$ whenever either $\star = \triangleleft$ and $i' = i{-}1$ (move left), or $\star = \triangleright$ and $i' = i{+}1$ (move right), with $0 \leqslant i' \leqslant |u|{+}1$. A run is a sequence of configurations $(q_1, i_1) \to \cdots \to (q_n, i_n)$. Accepting runs are those that begin in $(q_0, 0)$ and end in a configuration of the form $(q, |u|{+}1)$ with $q \in F$ (and never visit such a configuration before).

The partial function $f : A^* \rightharpoonup B^*$ computed by the two-way transducer $\mathscr{T}$ is defined as follows: for $u \in A^*$, if there exists an accepting run on $\vdash u \dashv$, then it is unique, and $f(u)$ is defined as $\lambda(q_1, (\vdash u \dashv)[i_1]) \cdots \lambda(q_n, (\vdash u \dashv)[i_n]) \in B^*$. The class of functions computed by two-way transducers is called **regular functions**.

Example 2.2. Let $\widetilde{u}$ be the mirror image of $u \in A^*$. Let $\# \notin A$ be a fresh symbol. The function $\mathsf{map\text{-}reverse} : u_1 \# \cdots \# u_n \mapsto \widetilde{u_1} \# \cdots \# \widetilde{u_n}$ can be computed by a two-way transducer, that reads each factor u_j from right to left.

It is well-known that the domain of a regular function is always a **regular language** (see e.g. [18]). From now on, we assume without losing generalities that our two-way transducers only compute total functions (in other words, they have exactly one accepting run on each $\vdash u \dashv$). Furthermore, we assume that $\lambda(q, \vdash) = \lambda(q, \dashv) = \varepsilon$ for all $q \in Q$ (we only lose generality for the image of ε).

In the rest of this section, $\mathscr{T}$ denotes a two-way transducer with input alphabet A, output alphabet B and output function λ. Now, we define the **crossing sequence** in a position $1 \leqslant i \leqslant |u|$ of input $\vdash u \dashv$. Intuitively, it regroups the states of the accepting run which are visited in this position.

Definition 2.3. *Let $u \in A^*$ and $1 \leqslant i \leqslant |u|$. Let $(q_1, i_1) \to \cdots \to (q_n, i_n)$ be the accepting run of $\mathscr{T}$ on $\vdash u \dashv$. The **crossing sequence** of $\mathscr{T}$ in i, denoted $\mathsf{cross}^u_{\mathscr{T}}(i)$, is defined as the sequence $(q_j)_{1 \leqslant j \leqslant n \text{ and } i_j = i}$.*

If $\mu : A^* \to \mathbb{M}$ is a monoid morphism, we say that any $m, m' \in \mathbb{M}$ and $a \in A$ define a **μ-context** that we denote by $m[\![a]\!]m'$. It is well-known that the crossing sequence in a position of the input only depends on the context of this position, for a well-chosen monoid, as claimed in proposition 2.4 (see e.g. [7]).

Proposition 2.4. *One can build a finite monoid $\mathbb{T}$ and a monoid morphism $\mu : A^* \to \mathbb{T}$, called the **transition morphism** of $\mathscr{T}$, such that for all $u \in A^*$ and $1 \leqslant i \leqslant |u|$, $\mathsf{cross}^u_{\mathscr{T}}(i)$ only depends on $\mu(u[1{:}i{-}1]), u[i]$ and $\mu(u[i{+}1{:}|u|])$. Thus we denote it $\mathsf{cross}_{\mathscr{T}}(\mu(u[1{:}i{-}1])[\![u[i]]\!]\mu(u[i{+}1{:}|u|]))$.*

Finally, let us define "the output produced below position i".

Definition 2.5. *Let $u \in A^*$ and $1 \leqslant i \leqslant |u|$ and $q_1 \cdots q_n := \mathsf{cross}^u_{\mathscr{T}}(i)$. We define the **production** of $\mathscr{T}$ in i, denoted $\mathsf{prod}^u_{\mathscr{T}}(i)$, as $\lambda(q_1, u[i]) \cdots \lambda(q_n, u[i])$.*

By proposition 2.4, it also makes sense to define $\mathsf{prod}_{\mathscr{T}}(m[\![a]\!]m') \in B^*$ to be $\mathsf{prod}^u_{\mathscr{T}}(i)$ whenever $m = \mu(u[1{:}i{-}1])$, $m' = \mu(u[i{+}1{:}|u|])$ and $a = u[i]$.

3 Blind and last pebble transducers

Now, we are ready to define formally the models of blind pebble transducers and last pebble transducers. Intuitively, they correspond to two-way transducers which make a tree of recursive calls to other two-way transducers.

Definition 3.1 (Blind pebble transducer [17]). *For $k \geqslant 1$, a **blind k-pebble transducer** with input alphabet A and output alphabet B is:*
- *if $k = 1$, a two-way transducer with input alphabet A and output B;*
- *if $k \geqslant 2$, a tree $\mathscr{T}\langle \mathscr{B}_1, \cdots, \mathscr{B}_p \rangle$ where the subtrees $\mathscr{B}_1, \ldots, \mathscr{B}_p$ are blind $(k{-}1)$-pebble transducers with input A and output B; and the root label $\mathscr{T}$ is a two-way transducer with input A and output alphabet $\{\mathscr{B}_1, \ldots, \mathscr{B}_p\}$.*

The (total) function $f : A^* \to B^*$ computed by the blind k-pebble transducer of definition 3.1 is built in a recursive fashion, as follows:
- for $k = 1$, f is the function computed by the two-way transducer;
- for $k \geqslant 2$, let $u \in A^*$ and $(q_1, i_1) \to \cdots \to (q_n, i_n)$ be the accepting run of $\mathscr{T} = (A, B, Q, q_0, F, \delta, \lambda)$ on $\vdash u \dashv$. For all $1 \leqslant j \leqslant n$, let $f_j : A^* \to B^*$ be the concatenation of the functions recursively computed by the sequence $\lambda(q_j, (\vdash u \dashv)[i_j]) \in \{\mathscr{B}_1, \ldots, \mathscr{B}_p\}^*$. Then $f(u) \coloneqq f_1(u) \cdots f_n(u)$.

The behavior of a blind 3-pebble transducer is depicted in fig. 2.

Example 3.2. The function $\mathsf{unmarked\text{-}square} : A^* \to A^* \uplus \{\#\}, u \mapsto (u\#)^{|u|}$ can be computed by a blind 2-pebble transducer. This machine has shape $\mathscr{T}\langle \mathscr{T}' \rangle$: $\mathscr{T}$ calls $\mathscr{T}'$ on each position $1 \leqslant i \leqslant |u|$ of its input u, and $\mathscr{T}'$ outputs $u\#$.

The class of functions computed by a blind k-pebble transducer for some $k \geqslant 1$ is called **polyblind functions** [10]. They form a strict subclass of polyregular functions [8,10,17] which is closed under composition [17, Theorem 6.1].

Now, let us define last pebble transducers. They corresponds to blind pebble transducers enhanced with the ability to mark the current position of the input when doing a recursive call. Formally, this position is underlined and we define $u{\bullet}i \coloneqq u[1] \cdots u[i{-}1]\underline{u[i]}u[i{+}1] \cdots u[|u|]$ for $u \in A^*$ and $1 \leqslant i \leqslant |u|$.

Definition 3.3 (Last pebble transducer [13]). *For $k \geqslant 1$, a **last k-pebble transducer** with input alphabet A and output alphabet B is:*
- *if $k = 1$, a two-way transducer with input alphabet $A \uplus \underline{A}$ and output B;*
- *if $k \geqslant 2$, a tree $\mathscr{T}\langle \mathscr{L}_1, \cdots, \mathscr{L}_p \rangle$ where the subtrees $\mathscr{L}_1, \ldots, \mathscr{L}_p$ are last $(k{-}1)$-pebble transducers with input A and output B; and the root label $\mathscr{T}$ is a two-way transducer with input $A \uplus \underline{A}$ and output alphabet $\{\mathscr{L}_1, \ldots, \mathscr{L}_p\}$.*

The (total) function $f : (A \uplus \underline{A})^* \to B^*$ computed by the last k-pebble transducer of definition 3.3 is defined in a recursive fashion, as follows:
- for $k = 1$, f is the function computed by the two-way transducer;

– for $k \geqslant 2$, let $u \in A^*$ and $(q_1, i_1) \to \cdots \to (q_n, i_n)$ be the accepting run of $\mathscr{T} = (A \uplus \underline{A}, B, Q, q_0, F, \delta, \lambda)$ on $\vdash u \dashv$. For all $1 \leqslant j \leqslant n$, let $f_j : A^* \to B^*$ be the concatenation of the functions recursively computed by $\lambda(q_j, (\vdash u \dashv)[i_j]) \in \{\mathscr{L}_1, \ldots, \mathscr{L}_p\}^*$. Let $\tau : (A \uplus \underline{A})^* \to A^*$ be the morphism which erases the underlining (i.e. $\tau(\underline{a}) = a$), then $f(u) \coloneqq f_1(\tau(u) \bullet i_1) \cdots f_n(\tau(u) \bullet i_n)$.

The behavior of a last 3-pebble transducer is depicted in fig. 3. Observe that our definition builds a function of type $(A \uplus \underline{A})^* \to B^*$, but we shall in fact consider its restriction to A^* (the marks are only used within the induction step).

Example 3.4 ([1]). The function $\mathsf{square} : u \mapsto (u \bullet 1) \# \cdots (u \bullet |u|) \#$ can be computed by a last 2-pebble transducer, which successively marks and makes recursive calls in positions $1, 2$, etc. However this function is not polyblind [17].

We are ready to state our main result. Its proof goes over sections 4 to 6.

Theorem 3.5 (Minimization of the recursion height). *Let $1 \leqslant \ell \leqslant k$. Let $f : A^* \to B^*$ be computed by a blind k-pebble transducer (resp. by a last k-pebble transducer). Then f can be computed by a blind ℓ-pebble transducer (resp. by a last ℓ-pebble transducer) if and only if $|f(u)| = \mathcal{O}(|u|^\ell)$.*
This property is decidable and the construction is effective.

As an easy consequence, the class of functions computed by last pebble transducers form a strict subclass of the polyregular functions (because theorem 3.5 does not hold for the full model of pebble transducers [3, Theorem 6.3]) and therefore it is not closed under composition (because any polyregular function can be obtained as a composition of regular functions and squares [1]).

Even if a (non-effective) theorem 3.5 was already known for blind pebble transducers [17, Theorem 7.1], we shall first present our proof of this case. Indeed, it is a new proof (relying on factorization forests) which is simpler than the original one. Furthermore, understanding the techniques used is a key step for understanding the proof for last pebble transducers presented afterwards.

4 Factorization forests

In this section, we introduce the key tool of factorization forests. Given a monoid morphism $\mu : A^* \to \mathbb{M}$ and $u \in A^*$, a μ-factorization forest of u is an unranked tree structure defined as follows. We use the brackets $\langle \cdots \rangle$ to build a tree.

Definition 4.1 (Factorization forest [19]). *Given a morphism $\mu : A^* \to \mathbb{M}$ and $u \in A^*$, we say that $\mathcal{F}$ is a μ-**forest** of u if:*
– *either $u = \varepsilon$ and $\mathcal{F} = \varepsilon$; or $u = \langle a \rangle \in A$ and $\mathcal{F} = a$;*
– *or $\mathcal{F} = \langle \mathcal{F}_1, \cdots, \mathcal{F}_n \rangle$, $u = u_1 \cdots u_n$, for all $1 \leqslant i \leqslant n$, $\mathcal{F}_i$ is a μ-forest of $u_i \in A^+$, and if $n \geqslant 3$ then $\mu(u) = \mu(u_1) = \cdots = \mu(u_n)$ is idempotent.*

We use the standard tree vocabulary of height, child, sibling, descendant and ancestor (a node being itself one of its ancestors/descendants), etc. We denote by $\mathsf{Nodes}^{\mathcal{F}}$ the set of nodes of $\mathcal{F}$. In order to simplify the statements, we identify

a node $\mathsf{t} \in \mathsf{Nodes}^{\mathcal{F}}$ with the subtree rooted in this node. Thus $\mathsf{Nodes}^{\mathcal{F}}$ can also be seen as the set of subtrees of $\mathcal{F}$, and $\mathcal{F} \in \mathsf{Nodes}^{\mathcal{F}}$. We say that a node is **idempotent** if it has at least 3 children. We denote by $\mathsf{Forests}_\mu(u)$ (resp. $\mathsf{Forests}^d_\mu(u)$) the set of μ-forests of $u \in A^*$ (resp. μ-forests of $u \in A^*$ of height at most d). We write $\mathsf{Forests}_\mu$ and $\mathsf{Forests}^d_\mu$ of all forests (of any word).

A μ-forest of $u \in A^*$ can also be seen as "the word u with brackets" in definition 4.1. Therefore $\mathsf{Forests}_\mu$ can be seen as a language over $\widehat{A} := A \uplus \{\langle, \rangle\}$. In this setting, it is well-known that μ-forests of bounded height can effectively be computed by a **rational function**, i.e. a particular case of regular function that can be computed by a non-deterministic one-way transducer (see e.g. [8]).

Theorem 4.2 (Simon [19,6]). *Given a morphism $\mu : A^* \to \mathbb{M}$ into a finite monoid $\mathbb{M}$, one can effectively build a rational function $\mathsf{forest}_\mu : A^* \to (\widehat{A})^*$ such that for all $u \in A^*$, $\mathsf{forest}_\mu(u) \in \mathsf{Forests}^{3|\mathbb{M}|}_\mu(u)$.*

Building μ-forests of bounded height is especially useful for us, since it enables to decompose any word in a somehow bounded way. This decomposition will be guided by the following definitions, that have been introduced in [8,10]. First, we define iterable nodes as the middle children of idempotent nodes.

Definition 4.3. *Let $\mathcal{F} \in \mathsf{Forests}_\mu(u)$. Its **iterable nodes**, denoted $\mathsf{Iter}^{\mathcal{F}}$, are:*
- *if $\mathcal{F} = \langle a \rangle \in A$ or $\mathcal{F} = \varepsilon$, then $\mathsf{Iter}^{\mathcal{F}} := \varnothing$;*
- *otherwise if $\mathcal{F} = \langle \mathcal{F}_1, \cdots, \mathcal{F}_n \rangle$, then:*

$$\mathsf{Iter}^{\mathcal{F}} := \{\mathcal{F}_i : 2 \leqslant i \leqslant n-1\} \cup \bigcup_{1 \leqslant i \leqslant n} \mathsf{Iter}^{\mathcal{F}_i}.$$

Now, we define the notion of skeleton of a node t, which contains all the descendants of t except those which are iterable.

Definition 4.4 (Skeleton, frontier). *Let $\mathcal{F} \in \mathsf{Forests}_\mu(u)$, $\mathsf{t} \in \mathsf{Nodes}^{\mathcal{F}}$, we define the **skeleton** of t, denoted $\mathsf{Skel}^{\mathcal{F}}(\mathsf{t})$, by:*
- *if $\mathsf{t} = \langle a \rangle \in A$ is a leaf, then $\mathsf{Skel}^{\mathcal{F}}(\mathsf{t}) := \{\mathsf{t}\}$;*
- *otherwise if $\mathsf{t} = \langle \mathcal{F}_1, \cdots, \mathcal{F}_n \rangle$, then $\mathsf{Skel}^{\mathcal{F}}(\mathsf{t}) := \{\mathsf{t}\} \cup \mathsf{Skel}^{\mathcal{F}}(\mathcal{F}_1) \cup \mathsf{Skel}^{\mathcal{F}}(\mathcal{F}_n)$.*

*The **frontier** of t is the set $\mathsf{Fr}^{\mathcal{F}}(\mathsf{t}) \subseteq [1{:}|u|]$ containing the positions of u which belong to $\mathsf{Skel}^{\mathcal{F}}(\mathsf{t})$ (when seen as leaves of the μ-forest $\mathcal{F}$ over u).*

Example 4.5. Let $\mathbb{M} := (\{-1, 1, 0\}, \times)$ and $\mu : \mathbb{M}^* \to \mathbb{M}$ the product. A μ-forest $\mathcal{F}$ of the word $(-1)(-1)0(-1)000000$ is depicted in Figure 4. Double lines denote idempotent nodes. The set of blue nodes is the skeleton of the topmost blue node.

It is easy to observe that for $\mathcal{F} \in \mathsf{Forests}^d_\mu(u)$, the size of a skeleton, or of a frontier, is bounded independently from $\mathcal{F}$. Furthermore, the set of skeletons $\{\mathsf{Skel}^{\mathcal{F}}(\mathsf{t}) : \mathsf{t} \in \mathsf{Iter}^{\mathcal{F}} \cup \{\mathcal{F}\}\}$ is a partition of $\mathsf{Nodes}^{\mathcal{F}}$ [8, Lemma 33]. As a consequence, the set of frontiers $\{\mathsf{Fr}^{\mathcal{F}}(\mathsf{t}) : \mathsf{t} \in \mathsf{Iter}^{\mathcal{F}} \cup \{\mathcal{F}\}\}$ is a partition of $[1{:}|u|]$. Given a position $1 \leqslant i \leqslant |u|$, we can thus define the **origin** of i in $\mathcal{F}$, denoted $\mathsf{origin}^{\mathcal{F}}(i)$, as the unique $\mathsf{t} \in \mathsf{Iter}^{\mathcal{F}} \cup \{\mathcal{F}\}$ such that $i \in \mathsf{Fr}^{\mathcal{F}}(\mathsf{t})$.

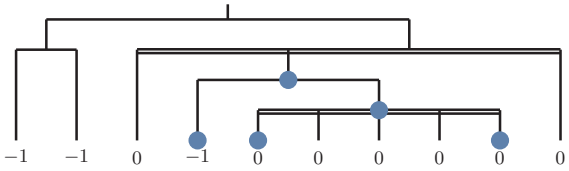


Figure 4: $\mathcal{F} \in \mathsf{Forests}_\mu((-1)(-1)0(-1)000000)$ and a skeleton.

Definition 4.6 (Observation). *Let* $\mathcal{F} \in \mathsf{Forests}_\mu$ *and* $\mathfrak{t}, \mathfrak{t}' \in \mathsf{Nodes}^{\mathcal{F}}$. *We say that* $\mathfrak{t} \in \mathsf{Nodes}^{\mathcal{F}}$ ***observes*** $\mathfrak{t}' \in \mathsf{Nodes}^{\mathcal{F}}$ *if either* $\mathfrak{t}'$ *is an ancestor of* $\mathfrak{t}$, *or* $\mathfrak{t}'$ *is the immediate right or left sibling of an ancestor of* $\mathfrak{t}$.

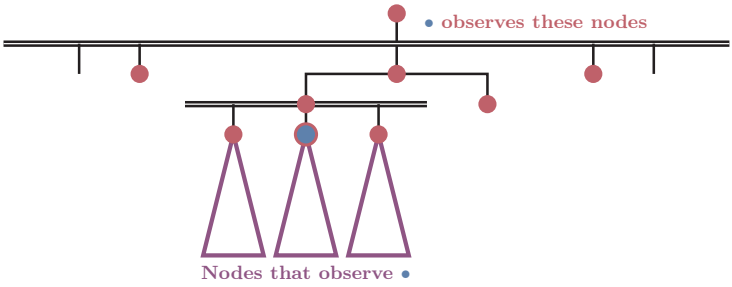


Figure 5: Nodes that observe • and that • observes

The intuition behind the notion of observation (which is *not* symmetrical) is depicted in fig. 5. Note that in a forest of bounded height, the number of nodes that some $\mathfrak{t}$ observes is bounded. This will be a key argument in the following. We say that $\mathfrak{t}$ and $\mathfrak{t}'$ are **dependent** if either $\mathfrak{t}$ observes $\mathfrak{t}'$ or the converse. Given $\mathcal{F}$, we can translate these notions to the positions of u: we say that i **observes** (resp. **depends** on) i' if $\mathsf{origin}^{\mathcal{F}}(i)$ observes (resp. depends on) $\mathsf{origin}^{\mathcal{F}}(i')$.

5 Height minimization of blind pebble transducers

In this section, we show theorem 3.5 for blind pebble transducers. We say that a two-way transducer $\mathscr{T}$ is a **submachine** of a blind pebble transducer $\mathscr{B}$ if $\mathscr{T}$ labels a node in the tree description of $\mathscr{B}$. If $\mathscr{B} = \mathscr{T}\langle\mathscr{B}_1, \ldots, \mathscr{B}_n\rangle$, we say that the submachine $\mathscr{T}$ is the **head** of $\mathscr{B}$. We let the **transition morphism** of $\mathscr{B}$ be the cartesian product of all the transition morphisms of all the submachines of $\mathscr{B}$. Observe that it makes sense to consider the production of a submachine $\mathscr{T}$ in a context defined using the transition morphism of $\mathscr{B}$.

5.1 Pumpability

We first give a sufficient condition, named pumpability, for a blind k-pebble transducer to compute a function f such that $|f(u)| \neq \mathcal{O}(|u|^{k-1})$. The behavior of a pumpable blind 2-pebble transducer is depicted in fig. 6 over a well-chosen input: it has a factor in which the head $\mathscr{T}_1$ calls a submachine $\mathscr{T}_2$, and a factor in which $\mathscr{T}_2$ produces a non-empty output. Furthermore both factors can be iterated without destroying the runs of these machines (due to idempotents).

Definition 5.1. *Let $\mathscr{B}$ be a blind k-pebble transducer whose transition morphism is $\mu : A^* \to \mathbb{T}$. We say that the transducer $\mathscr{B}$ is **pumpable** if there exists:*
- *submachines $\mathscr{T}_1, \dots, \mathscr{T}_k$ of $\mathscr{B}$, such that $\mathscr{T}_1$ is the head of $\mathscr{B}$;*
- *$m_0, \dots, m_k, \ell_1, \dots, \ell_k, r_1, \dots, r_k \in \mu(A^*)$;*
- *$a_1, \dots, a_k \in A$ such that for all $1 \leqslant j \leqslant k$, $e_j \coloneqq \ell_j \mu(a_j) r_j$ is an idempotent;*
- *a permutation $\sigma : [1{:}k] \to [1{:}k]$;*

such that if $\mathcal{M}_i^j \coloneqq m_i e_{i+1} m_{i+1} \cdots e_j m_j$ for all $0 \leqslant i \leqslant j \leqslant k$, and if we define the following context for all $1 \leqslant j \leqslant k$:

$$\mathcal{C}_j \coloneqq \mathcal{M}_0^{\sigma(j)-1} e_{\sigma(j)} \ell_{\sigma(j)} [\![a_{\sigma(j)}]\!] r_{\sigma(j)} e_{\sigma(j)} \mathcal{M}_{\sigma(j)}^k$$

then for all $1 \leqslant j \leqslant k-1$, $|\mathsf{prod}_{\mathscr{T}_j}(\mathcal{C}_j)|_{\mathscr{T}_{j+1}} \neq 0$, and $\mathsf{prod}_{\mathscr{T}_k}(\mathcal{C}_k) \neq \varepsilon$.

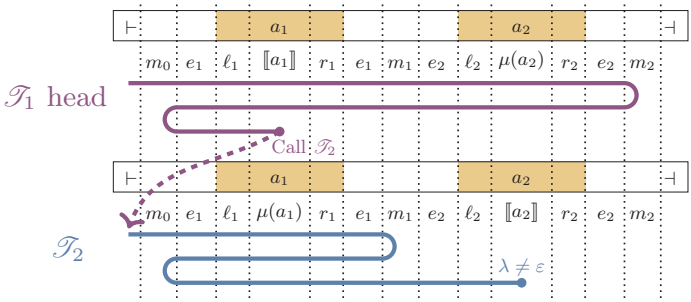


Figure 6: Pumpability in a blind 2-pebble transducer.

Lemma 5.2 follows by choosing inverse images in A^* for the m_i, ℓ_i and r_i.

Lemma 5.2. *Let f be computed by a pumpable blind k-pebble transducer. There exists words $v_0, \dots, v_k, u_1, \dots, u_k$ such that $|f(v_0 u_1^X \cdots u_k^X v_k)| = \Theta(X^k)$.*

Now, we use pumpability as a key ingredient for showing theorem 3.5, which directly follows by induction from the more precise theorem 5.3.

Theorem 5.3 (Removing one layer). *Let $k \geqslant 2$ and $f : A^* \to B^*$ be computed by a blind k-pebble transducer $\mathscr{B}$. The following are equivalent:*
1. *$|f(u)| = \mathcal{O}(|u|^{k-1})$;*

2. $\mathscr{B}$ is not pumpable;
3. f can be computed by a blind $(k-1)$-pebble transducer.
Furthermore, this property is decidable and the construction is effective.

Proof. Item 3 $\Rightarrow$ item 1 is obvious. Item 1 $\Rightarrow$ item 2 is lemma 5.2. Furthermore, pumpability can be tested by an enumeration of $\mu(A^*)$ and A. It remains to show item 2 $\Rightarrow$ item 3 (in an effective fashion): this is the purpose of section 5.2.

5.2 Algorithm for removing a recursion layer

Let $k \geqslant 2$ and $\mathscr{U}$ be a blind k-pebble transducer that is not pumpable, and that computes $f : A^* \to B^*$. We build a blind $(k-1)$-pebble transducer $\overline{\mathscr{U}}$ for f.

Let $\mu : A^* \to \mathbb{T}$ be the transition morphism of $\mathscr{U}$. We shall consider that, on input $u \in A^*$, the submachines of $\overline{\mathscr{U}}$ can in fact use $\mathsf{forest}_\mu(u) \subseteq (\widehat{A})^*$ as input. Indeed forest_μ is a rational function (by theorem 4.2), hence its information can be recovered by using a **lookaround**. Informally, the lookaround feature enables a two-way transducer to chose its transitions not only depending on its current state and current letter $u[i]$ in position $1 \leqslant i \leqslant |u|$, but also on a regular property of the prefix $u[1{:}i{-}1]$ and the suffix $u[i{+}1{:}|u|]$. It is well-known that given a two-way transducer $\mathscr{T}$ with lookarounds, one can build an equivalent $\mathscr{T}'$ that does not have this feature (see e.g. [15,12]). Furthermore, even if the accepting runs of $\mathscr{T}$ and $\mathscr{T}'$ may differ, they produce the same outputs from the same positions (this observation will be critical for last pebble transducers, in order to ensure that the marked positions of the recursive calls will be preserved).

Now, we describe the two-way transducers that are the submachines of $\overline{\mathscr{U}}$. First, it has submachines $\mathsf{old}\text{-}\mathscr{T}$ for $\mathscr{T}$ a submachine of $\mathscr{U}$, which are described in algorithm 1. Intuitively, $\mathsf{old}\text{-}\mathscr{T}$ is just a copy of $\mathscr{T}$. It is clear that if $\mathscr{T}$ is a submachine of $\mathscr{U}$, then $\mathsf{old}\text{-}\mathscr{T}(u)$ is the concatenation of the outputs produced by (the recursive calls of) $\mathscr{T}$ along its accepting run on $\vdash u \dashv$.

Algorithm 1: Submachines that behave as the original ones

```
1  Submachine old-𝒯(u)
2      ρ := accepting run of 𝒯 over ⊢u⊣; λ := output function of 𝒯;
3      for (q, i) ∈ ρ do
4          if 𝒯 is a leaf of 𝒰 then
5              Output λ(q, (⊢u⊣)[i]); /* 𝒯 has output in B*;          */
6          else
7              for ℬ' ∈ λ(q, (⊢u⊣)[i]) do
8                  𝒯' := head of ℬ';
9                  Call old-𝒯'(u); /* 𝒯 makes recursive calls;      */
10             end
11         end
12     end
```

$\overline{\mathscr{U}}$ also has submachines accelerate-$\mathscr{T}$ for $\mathscr{T}$ a submachine of $\mathscr{U}$, which are described in algorithm 2. Intuitively, accelerate-$\mathscr{T}$ simulates $\mathscr{T}$ while trying to inline recursive calls in its own run. More precisely, let $u \in A^*$ be the input and $\mathcal{F} \coloneqq \mathsf{forest}_\mu(u)$. If $\mathscr{T}$ calls $\mathscr{B}'$ in $1 \leqslant i \leqslant |u|$ that belongs to the frontier of the root node $\mathcal{F}$ of $\mathcal{F}$, then accelerate-$\mathscr{T}$ inlines the behavior of the head of $\mathscr{B}'$. Otherwise it makes a recursive call, except if $\mathscr{B}'$ is a leaf of $\mathscr{U}$. Hence if $\mathscr{T}$ is a submachine of $\mathscr{U}$ which is not a leaf, accelerate-$\mathscr{T}(u)$ is the concatenation of the outputs produced by the calls of $\mathscr{T}$ along its accepting run.

Algorithm 2: Submachines that try to simulate their recursive calls

```
1  Submachine accelerate-𝒯 (u)
2      /* 𝒯 is not a leaf of 𝒰 (i.e. it makes calls);                    */
3      ρ ≔ accepting run of 𝒯 over ⊢u⊣; ℱ ≔ forest_μ(u); λ ≔ output fun. of 𝒯;
4      for (q, i) ∈ ρ do
5          for ℬ′ ∈ λ(q, (⊢u⊣)[i]) do
6              𝒯′ ≔ head of ℬ′;
7              if i ∈ Fr^ℱ(ℱ) then
8                  /* We can inline the call since |Fr^ℱ(ℱ)| is bounded;  */
9                  Inline the code of old-𝒯′ (u) /* (see explanations);   */
10             else if ℬ′ is a leaf of 𝒰 then
11                 /* Then ℬ′ = 𝒯′ and we can inline the call because the
                      output of 𝒯′ on input u is bounded;                 */
12                 Inline the code of old-𝒯′ (u) /* (see explanations);   */
13             else
14                 /* It is not possible to inline the call to ℬ′, so we
                      make a recursive call;                              */
15                 Call accelerate-𝒯′ (u);
16             end
17         end
18     end
```

Finally, the transducer $\overline{\mathscr{U}}$ is obtained by defining accelerate-$\mathscr{T}$ to be its head, where $\mathscr{T}$ is the head of $\mathscr{U}$. Furthermore, we remove the submachines old-$\mathscr{T}$ or accelerate-$\mathscr{T}$ which are never called. Observe that $\overline{\mathscr{U}}$ indeed computes the function f. Furthermore, we observe that $\overline{\mathscr{U}}$ has recursion height (i.e. the number of nested **Call** instructions, plus 1 for the head) $k-1$, since each inlining of lines 9, 10 and 12 in algorithm 2 removes exactly one recursion layer of $\mathscr{U}$.

It remains to justify that each accelerate-$\mathscr{T}$ can be implemented by a two-way transducer (i.e. with lookarounds but a bounded memory). We represent variable i by the current position of the transducer. Since it has access to $\mathcal{F}$, the lookaround can be used to check whether $i \in \mathsf{Fr}^{\mathcal{F}}(\mathcal{F})$ or not (since the size of $\mathsf{Fr}^{\mathcal{F}}(\mathcal{F})$ is bounded). It remains to explain how the inlinings are performed:

– if $i \in \mathsf{Fr}^{\mathcal{F}}(\mathcal{F})$, the two-way transducer inlines old-$\mathscr{T}'$ by executing the same moves and calls as $\mathscr{T}'$ does. Once its computation is ended, it has to go back

to position i. This is indeed possible since belonging to $\mathsf{Fr}^{\mathcal{F}}(\mathcal{F})$ is a property that can be detected by using the lookaround, hence the machine only needs to remember that i was the ℓ-th position of $\mathsf{Fr}^{\mathcal{F}}(\mathcal{F})$ (ℓ being bounded);
– else if $\mathscr{B}' = \mathscr{T}'$ is a blind 1-pebble transducer, we produce the output of $\mathscr{T}'$ without moving. This is possible since for all $i' \notin \mathsf{Fr}^{\mathcal{F}}(\mathcal{F})$, $\mathsf{prod}^u_{\mathscr{T}'}(i') = \varepsilon$ (hence the output of $\mathscr{T}'$ on u is bounded, and its value can be determined without moving, just by using the lookaround). Indeed, if $\mathsf{prod}^u_{\mathscr{T}'}(i') \neq \varepsilon$ for such an $i' \notin \mathsf{Fr}^{\mathcal{F}}(\mathcal{F})$ when reaching line 12 of algorithm 2, then the conditions of lemma 5.4 hold, which yields a contradiction. This lemma is the key argument of this proof, relying on the non-pumpability of $\mathscr{U}$.

Lemma 5.4 (Key lemma). *Let $u \in A^*$ and $\mathcal{F} \in \mathsf{Forests}_\mu(u)$. Assume that there exists a sequence $\mathscr{T}_1, \dots, \mathscr{T}_k$ of submachines of $\mathscr{U}$ and a sequence of positions $1 \leqslant i_1, \dots, i_k \leqslant |u|$ such that:*
– *$\mathscr{T}_1$ is the head of $\mathscr{U}$;*
– *for all $1 \leqslant j \leqslant k-1$, $|\mathsf{prod}^u_{\mathscr{T}_j}(i_j)|_{\mathscr{T}_{j+1}} \neq 0$ and $\mathsf{prod}^u_{\mathscr{T}_k}(i_k) \neq \varepsilon$;*
– *for all $1 \leqslant j \leqslant k$, $i_j \notin \mathsf{Fr}^{\mathcal{F}}(\mathcal{F})$ (i.e. $\mathsf{origin}^{\mathcal{F}}(i_j) \in \mathsf{Iter}^{\mathcal{F}}$).*
Then $\mathscr{B}$ is pumpable.

Proof (idea). We first observe that pumpability follows as soon as the nodes $\mathsf{origin}^{\mathcal{F}}(i_j)$ are pairwise independent. We then show that this independence condition can always be obtained, up to duplicating some iterable subtrees of $\mathcal{F}$ (and some factors of u), because the behavior of a submachine in a blind pebble transducer does not depend on the positions of the above recursive calls.

6 Height minimization of last pebble transducers

In this section, we show theorem 3.5 for last pebble transducers. The notions of **submachine**, **head** and **transition morphism** for a last pebble transducer are defined as in section 5. The transition morphism is now defined over $(A \uplus \underline{A})^*$.

6.1 Pumpability

The sketch of the proof is similar to section 5. We first give an equivalent of pumpability for last pebble transducers. The intuition behind this notion is depicted in fig. 7. The formal definition is however more cumbersome, since we need to keep track of the fact that the calling position is marked.

Definition 6.1. *Let $\mathscr{L}$ be a last k-pebble transducer whose transition morphism is $\mu : (A \cup \underline{A})^* \to \mathbb{T}$. We say that the transducer $\mathscr{L}$ is **pumpable** if there exists:*
– *submachines $\mathscr{T}_1, \dots, \mathscr{T}_k$ of $\mathscr{L}$, such that $\mathscr{T}_1$ is the head of $\mathscr{L}$;*
– *$m_0, \dots, m_k, \ell_1, \dots, \ell_k, r_1, \dots, r_k \in \mu(A^*)$;*
– *$a_1, \dots, a_k \in A$ such that for all $1 \leqslant j \leqslant k$, $e_j \coloneqq \ell_j \mu(a_j) r_j$ is idempotent;*
– *a permutation $\sigma : [1{:}k] \to [1{:}k]$;*

such that if we let $\mathcal{M}_i^j \coloneqq m_i e_{i+1} m_{i+1} \cdots e_j m_j$ *for all* $0 \leqslant i \leqslant j \leqslant k$, *and if we define the following context:*

$$\mathcal{C}_1 \coloneqq \mathcal{M}_0^{\sigma(1)-1} e_{\sigma(1)} \ell_{\sigma(1)} [\![a_{\sigma(1)}]\!] r_{\sigma(1)} e_{\sigma(1)} \mathcal{M}_{\sigma(1)}^k$$

and for all $1 \leqslant j \leqslant k-1$ *the context:*

$$\begin{aligned}
\mathcal{C}_{j+1} \coloneqq{}& \mathcal{M}_0^{\sigma(j)-1} e_{\sigma(j)} \ell_{\sigma(j)} \mu(\underline{a_{\sigma(j)}}) r_{\sigma(j)} e_{\sigma(j)} \mathcal{M}_{\sigma(j)}^{\sigma(j+1)-1} \\
& e_{\sigma(j+1)} \ell_{\sigma(j+1)} [\![a_{\sigma(j+1)}]\!] r_{\sigma(j+1)} e_{\sigma(j+1)} \mathcal{M}_{\sigma(j+1)}^k && \text{if } \sigma(j) < \sigma(j+1); \\
\mathcal{C}_{j+1} \coloneqq{}& \mathcal{M}_0^{\sigma(j)-1} e_{\sigma(j+1)} \ell_{\sigma(j+1)} [\![a_{\sigma(j+1)}]\!] r_{\sigma(j+1)} e_{\sigma(j+1)} \\
& \mathcal{M}_{\sigma(j+1)}^{\sigma(j)-1} e_{\sigma(j)} \ell_{\sigma(j)} \mu(\underline{a_{\sigma(j)}}) r_{\sigma(j)} e_{\sigma(j)} \mathcal{M}_{\sigma(j)}^k && \text{otherwise;}
\end{aligned}$$

then for all $1 \leqslant j \leqslant k-1$, $|\mathsf{prod}_{\mathscr{T}_j}(\mathcal{C}_j)|_{\mathscr{T}_{j+1}} \neq 0$, *and* $\mathsf{prod}_{\mathscr{T}_k}(\mathcal{C}_k) \neq \varepsilon$.

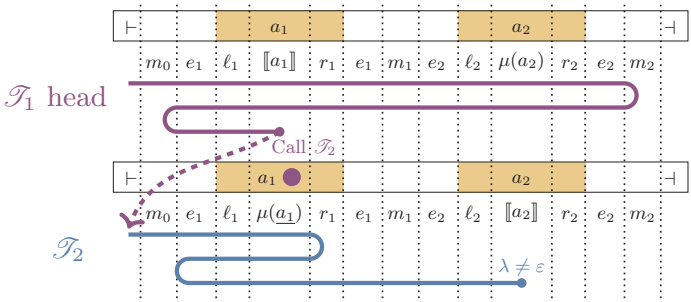


Figure 7: Pumpability in a last 2-pebble transducer.

We obtain lemma 6.2 by a proof which is similar to that of lemma 5.2.

Lemma 6.2. *Let* f *be computed by a pumpable last* k*-pebble transducer. There exists words* $v_0, \ldots, v_k, u_1, \ldots, u_k$ *such that* $|f(v_0 u_1^X \cdots u_k^X v_k)| = \Theta(X^k)$.

Theorem 6.3 (Removing one layer). *Let* $k \geqslant 2$ *and* $f : A^* \to B^*$ *be computed by a last* k*-pebble transducer* $\mathscr{L}$. *The following are equivalent:*

1. $|f(u)| = \mathcal{O}(|u|^{k-1})$;
2. $\mathscr{L}$ *is not pumpable;*
3. f *can be computed by a last* $(k-1)$*-pebble transducer.*

Furthermore, this property is decidable and the construction is effective.

Proof. Item 3 ⇒ item 1 is obvious. Item 1 ⇒ item 2 is lemma 6.2. Furthermore, pumpability can be tested by an enumeration of $\mu(A^*)$ and A. It remains to show item 2 ⇒ item 3 (in an effective fashion): this is the purpose of section 6.2.

6.2 Algorithm for removing a recursion layer

Let $k \geqslant 2$ and $\mathscr{U}$ be a last k-pebble transducer that is not pumpable, and that computes $f : A^* \to B^*$. We build a last $(k-1)$-pebble transducer $\overline{\mathscr{U}}$ for f. Let $\mu : (A \uplus \underline{A})^* \to \mathbb{T}$ be the transition morphism of $\mathscr{U}$. As before (using a lookaround), the submachines of $\overline{\mathscr{U}}$ have access to $\mathsf{forest}_\mu(u)$ on input $u \in A^*$.

Now, we describe the submachines of $\overline{\mathscr{U}}$. It has submachines old-$\mathscr{T}$-along-ρ for $\mathscr{T}$ a submachine of $\mathscr{U}$ and ρ a run of $\mathscr{T}$, which are described in algorithm 1. Intuitively, these machines mimics the behavior of $\mathscr{T}$ along the run ρ (which is not necessarily accepting) of $\mathscr{T}$ over $\vdash v \dashv$ with $v \in (A \uplus \underline{A})^*$.

Since they are indexed by a run ρ, it may seem that we create an infinite number of submachines, but it will not be the case. Indeed, a run ρ will be represented by its first configuration (q_1, i_1) and last configuration (q_n, i_n). This information is sufficient to simulate exactly the two-way moves of ρ, but there is still an unbounded information: the positions i_1 and i_n. In fact, the input will be of the form $v = u \bullet i$ and we shall guarantee that the i_1 and i_n can be detected by the lookaround if i is marked. Hence the run ρ will be represented in a bounded way, independently from the input v, and so that its first and last configurations can be detected by the lookaround of the submachine.

It follows from algorithm 3 that if $\mathscr{T}$ is a submachine of $\mathscr{U}$, then for all $v \in (A \cup \underline{A})^*$ and ρ run of $\mathscr{T}$ on $\vdash v \dashv$, old-$\mathscr{T}$-along-ρ (v) is the concatenation of the outputs produced by (the recursive calls of) $\mathscr{T}$ along ρ.

We also define a submachine normal-$\mathscr{T}$-along-ρ-pebble-**i** that is similar to old-$\mathscr{T}$-along-ρ, except that it ignores the mark of its input and acts as if it was in position i (as above for ρ, i will be encoded by a bounded information).

Algorithm 3: Submachines that behave like the original ones

```
1  Submachine old-𝒯-along-ρ(v)
2      /* v ∈ (A ⊎ A̲)*; ρ is a run of 𝒯 over ⊢v⊣;                    */
3      λ := output function of 𝒯;
4      for (q, i) ∈ ρ do
5          if 𝒯 is a leaf of 𝒰 then
6              Output λ(q, (⊢v⊣)[i]); /* 𝒯 has output in B*;          */
7          else
8              for 𝒯' ∈ λ(q, (⊢v⊣)[i]) do
9                  𝒯' := head of 𝒯'; ρ' := accepting run of 𝒯' on ⊢τ(v)•i⊣;
10                 Call old-𝒯'-along-ρ'(τ(v)•i); /* Recursive call;  */
11             end
12         end
13     end
14 Submachine normal-𝒯-along-ρ-pebble-i(v)
15     /* v ∈ (A ⊎ A̲)*; ρ is a run of 𝒯 over ⊢τ(v)•i⊣;               */
16     Simulate old-𝒯-along-ρ (τ(v)•i);
```

$\overline{\mathscr{U}}$ also has submachines accelerate-$\mathscr{T}$-along-ρ for $\mathscr{T}$ a submachine of $\mathscr{U}$, which are described in algorithm 4. Intuitively, accelerate-$\mathscr{T}$-along-ρ simulates $\mathscr{T}$ along ρ while trying to inline some recursive calls. Whenever it is in position i and needs to call recursively $\mathscr{L}'$ whose head is $\mathscr{T}'$, it first slices the accepting run ρ' of $\mathscr{T}'$ on $\vdash u \bullet i \dashv$, with respect to $\mathsf{forest}_\mu(u)$ and i, as explained in definition 6.4 and depicted in fig. 8. Intuitively, this operation splits ρ' into a bounded number of runs whose positions either all observe i, or i observes all of them, or none of these cases occur (the positions are either 0, $|u|{+}1$ or independent of i).

Definition 6.4 (Slicing). *Let $u \in A^*$, $\mathcal{F} \in \mathsf{Forests}_\mu(u)$ and $1 \leqslant i \leqslant |u|$. We let $\uparrow \mathbf{i}$ (resp. $\downarrow \mathbf{i}$) be the set of positions that i observes (resp. that observe i). Let $\rho = (q_1, i_1) \to \cdots \to (q_n, i_n)$ be a run of a two-way transducer $\mathscr{T}$ on $\vdash u \bullet i \dashv$. We build by induction a sequence $\ell_1, \ldots, \ell_{N+1}$ with $\ell_1 := 1$ and:*

- *if $\ell_j = n{+}1$ then $j := N$ and the process ends;*
- *else if $i_{\ell_j} \in \uparrow i$ (resp. $i_{\ell_j} \in \downarrow i \smallsetminus \uparrow i$, resp. $i_{\ell_j} \in [0{:}|u|{+}1] \smallsetminus (\uparrow i \cup \downarrow i)$), then ℓ_{j+1} is the largest index such that for all $\ell_j \leqslant \ell \leqslant \ell_{j+1}{-}1$, $i_\ell \in \uparrow i$ (resp. $i_\ell \in \downarrow i \smallsetminus \uparrow i$, resp. $i_\ell \in [0{:}|u|{+}1] \smallsetminus (\uparrow i \cup \downarrow i)$).*

*Finally the **slicing** of ρ ,with respect to $\mathcal{F}$ and i, is the sequence of runs $\rho_1, \ldots, \rho_N$ where $\rho_j := (q_{\ell_j}, i_{\ell_j}) \to (q_{\ell_j+1}, i_{\ell_j+1}) \to \cdots \to (q_{\ell_{j+1}-1}, i_{\ell_{j+1}-1})$.*

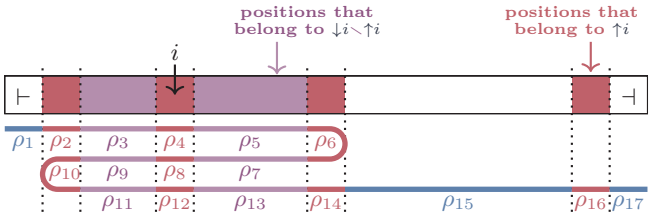


Figure 8: Slicing of a run ρ with respect to i and $\mathcal{F}$.

Now, let $\rho'_1, \ldots, \rho'_N$ be slicing of the run ρ' of $\mathscr{T}'$ on the input $u \bullet i$. For all $1 \leqslant j \leqslant N$, there are mainly two cases. Either the positions of ρ'_j all are in $\uparrow i$ or $\downarrow i$. In this case, accelerate-$\mathscr{T}$-along-ρ directly inlines old-$\mathscr{T}'$-along-ρ'_j within its own run (i.e. without making a recursive call). Otherwise, it makes a recursive call to accelerate-$\mathscr{T}'$-along-ρ'_j, except if $\mathscr{L}'$ is a leaf of $\mathscr{U}$ (thus $\mathscr{L}' = \mathscr{T}'$).

Finally, $\overline{\mathscr{U}}$ is described as follows: on input $u \in A^*$, its head is the submachine accelerate-$\mathscr{T}$-along-ρ (u), where $\mathscr{T}$ is the head of $\mathscr{U}$ and ρ is the accepting run of $\mathscr{T}$ on $\vdash u \dashv$ (represented by the bounded information that it is both initial and final). As before, we remove the submachines which are never called in $\overline{\mathscr{U}}$. Observe that we have created a machine with recursion height $k{-}1$ (because line 17 in algorithm 4 prevents from calling a k-th layer).

Let us justify that each accelerate-$\mathscr{T}$-along-ρ can indeed be implemented by a two-way transducer. First, let us observe that since $\mathcal{F}$ has bounded height, the number N of slices given in line 7 of algorithm 4 is bounded. Furthermore, we claim that the first and last positions of each ρ'_j belong to a given set of bounded size, which can be detected by a lookaround which has access to i. For the ρ'_j

Algorithm 4: Submachines that try to simulate their recursive calls

```
1  Submachine accelerate-𝒯-along-ρ (v)
2      /* 𝒯 is not a leaf of 𝒰 (i.e. it makes calls);                    */
3      /* v ∈ (A ⊎ A̲)*; ρ is a run of 𝒯 over ⊢v⊣;                        */
4      u ≔ τ(v); ℱ ≔ forest_μ(u); λ ≔ output function of 𝒯;
5      for (q, i) ∈ ρ do
6          for 𝒮′ ∈ λ(q, (⊢v⊣)[i]) do
7              𝒯′ ≔ head of 𝒮′; ρ′ ≔ accepting run of 𝒯′ over ⊢u•i⊣;
8              ρ′_1, ··· , ρ′_N ≔ slicing of ρ′ with respect to ℱ and i;
9              for j = 1 to N do
10                 (q_1, i_1) → ··· (q_n, i_n) ≔ ρ′_j
11                 if i_1, ..., i_n ∈ ↑ i then
12                     /* We inline the call because n is bounded;       */
13                     Inline the code of old-𝒯′-along-ρ′_j (u•i);
14                 else if i_1, ..., i_n ∈ ↓ i then
15                     /* We can inline the call because the positions
                          i_1, ..., i_n are "below" i in ℱ;              */
16                     Inline the code of old-𝒯′-along-ρ′_j (u•i);
17                 else if 𝒮′ is a leaf of 𝒰 then
18                     /* The output of 𝒮′ = 𝒯′ along ρ′_j is empty;     */
19                 else
20                     /* It is not possible to inline the call to 𝒮′, so
                          we make a recursive call;                      */
21                     Call accelerate-𝒯′-along-ρ′_j (u•i);
22                 end
23             end
24         end
25     end
```

whose positions are in $\uparrow i$, this is clear since $|\uparrow i|$ is bounded (because the frontier of any node is bounded). For $\downarrow i \smallsetminus \uparrow i$ we use lemma 6.5, which implies that this set is a bounded union of intervals. The last case is very similar.

Lemma 6.5. *Let $1 \leqslant i \leqslant |u|$, $\mathsf{t} \coloneqq \mathsf{origin}^{\mathcal{F}}(i)$ and t_1 (resp. t_2) be its immediate left (resp. right) sibling (they exist whenever $\mathsf{t} \in \mathsf{Iter}^{\mathcal{F}}$, i.e. here $\mathsf{t} \neq \mathcal{F}$). Then:*

$$\downarrow i \smallsetminus \uparrow i = [\min(\mathsf{Fr}^{\mathcal{F}}(\mathsf{t}_1)) : \max(\mathsf{Fr}^{\mathcal{F}}(\mathsf{t}_2))] \smallsetminus \{\mathsf{Fr}^{\mathcal{F}}(\mathsf{t}_1), \mathsf{Fr}^{\mathcal{F}}(\mathsf{t}), \mathsf{Fr}^{\mathcal{F}}(\mathsf{t}_2)\}.$$

This analysis justifies why each ρ'_j can be encoded in a bounded way. Now, we show how to implement the inlinings while using i as the current position:

– if $i_1, \ldots, i_n \in \uparrow i$, then n is bounded (because $|\uparrow i|$ is bounded). We can thus inline old-$\mathscr{T}'$-along-ρ'_j $(u\bullet i)$ while staying in position i. However, when $\mathscr{T}'$ calls some $\mathscr{L}''$ (of head $\mathscr{T}''$) on position i_ℓ, we would need to call old-$\mathscr{T}''$-along-$\rho''(u\bullet i_\ell)$ (where ρ'' is the accepting run of $\mathscr{T}''$ along $\vdash u\bullet i_\ell \dashv$). But we cannot do this operation, since we are in position i and not in i_ℓ. The solution is that the inlined code calls normal-$\mathscr{T}''$-along-ρ''-pebble-$i_\ell(u\bullet i)$

instead, which simulates an accepting run ρ'' of $\mathscr{T}$ on $u{\bullet}i_\ell$, even if its input is $u{\bullet}i$. Note that i_ℓ can be represented as a bounded information and recovered by a lookaround given $u{\bullet}i$ as input, since i observes i_ℓ;

– if $i_1, \dots, i_n \in \downarrow i \smallsetminus \uparrow i$, then the nodes $\mathsf{origin}^{\mathcal{F}}(i_1), \dots, \mathsf{origin}^{\mathcal{F}}(i_n)$ are roughly below $\mathsf{origin}^{\mathcal{F}}(i)$ in $\mathcal{F}$ (see fig. 5). We inline old-$\mathscr{T}'$-along-ρ'_j $(u{\bullet}i)$, by moving along $i_1, \dots, i_n$ as ρ'_j does. We can keep track of the height of $\mathsf{origin}^{\mathcal{F}}(i)$ above the current $\mathsf{origin}^{\mathcal{F}}(i_\ell)$ (it is a bounded information). With the lookaround, we can detect the end of ρ'_j, and go back to position i.

It remains to justify that $\overline{\mathscr{U}}$ is correct. For this, we only need to show that when it reaches line 18 in algorithm 4, the output of $\mathscr{T}'$ along ρ'_j is indeed empty. Otherwise, the conditions of lemma 6.6 would hold (since we never execute two successive recursive calls in dependent positions). It provides a contradiction.

Lemma 6.6 (Key lemma). *Let $u \in A^*$ and $\mathcal{F} \in \mathsf{Forests}_\mu(u)$. Assume that there exists a sequence $\mathscr{T}_1, \dots, \mathscr{T}_k$ of submachines of $\mathscr{U}$ and a sequence of positions $1 \leqslant i_1, \dots, i_k \leqslant |u|$ such that:*

– *$\mathscr{T}_1$ is the head of $\mathscr{U}$;*
– *$|\mathsf{prod}^{u}_{\mathscr{T}_1}(i_1)|_{\mathscr{T}_2} \neq 0$ and $\mathsf{prod}^{u{\bullet}i_{k-1}}_{\mathscr{T}_k}(i_k) \neq \varepsilon$;*
– *for all $2 \leqslant j \leqslant k{-}1$, $|\mathsf{prod}^{u{\bullet}i_{j-1}}_{\mathscr{T}_j}(i_j)|_{\mathscr{T}_{j+1}} \neq 0$;*
– *for all $1 \leqslant j \leqslant k{-}1$, $\mathsf{origin}^{\mathcal{F}}(i_j)$ and $\mathsf{origin}^{\mathcal{F}}(i_{j+1})$ are independent;*

Then $\mathscr{U}$ is pumpable.

Proof (idea). As for lemma 5.4, the key observation is that pumpability follows as soon as the nodes $\mathsf{origin}^{\mathcal{F}}(i_j)$ are pairwise independent. Furthermore, this condition can be obtained by duplicating some nodes in $\mathcal{F}$.

7 Making the two last pebbles visible

We can define a similar model to that of last k-pebble transducer, which sees the two last calling positions instead of only the previous one. Let us name this model a **last-last k-pebble transducer**. A very natural question is to know whether we can show an analog of theorem 3.5 for these machines.

Note that for $k = 1, 2$ and 3, a last-last k-pebble transducer is exactly the same as a k-pebble transducer. Hence the function inner-squaring of page 2 is such that $|\mathsf{inner\text{-}squaring}(u)| = \mathcal{O}(|u|^2)$ and can be computed by a last-last 3-pebble transducer, but it cannot be computed by a last-last 2-pebble transducer. It follows that the connection between minimal recursion height and growth of the output fails. However, this result is somehow artificial. Indeed, a last-last 2-pebble transducer is a degenerate case, since it can only see one last pebble. More interestingly, we show that the connection fails for arbitrary heights.

Theorem 7.1. *For all $k \geqslant 2$, there exists a function $f : A^* \to B^*$ such that $|f(u)| = \mathcal{O}(|u|^2)$ and that can be computed by a last-last $(2k{+}1)$-pebble transducer, but not by a last-last $2k$-pebble transducer.*

Proof (idea). We re-use a counterexample introduced by Bojańczyk in [2] to show a similar failure result for the model of k-pebble transducers.

8 Outlook

This paper somehow settles the discussion concerning the variants of pebble transducers for which the minimal recursion height only depends on the growth of the output. As soon as two marks are visible, the combinatorics of the output also has to be taken into account, hence minimizing the recursion height in this case (e.g. for last-last pebble transducers) seems hard with the current tools.

As observed in [13], one can extend last pebble transducers by allowing the recursion height to be unbounded (in the spirit of **marble transducers** [9]). This model enables to produce outputs whose size grows exponentially in the size of the input. A natural question is to know whether a function computed by this model, but whose output size is polynomial, can in fact be computed with a recursion stack of bounded height (i.e. by a last k-pebble transducer).

Acknowledgements. The author is grateful to Tito Nguyên for suggesting the study of the recursion height for last pebble transducers.

References

1. Bojańczyk, M.: Polyregular functions. arXiv preprint arXiv:1810.08760 (2018)
2. Bojańczyk, M.: The growth rate of polyregular functions. arXiv preprint arXiv:2212.11631 (2022)
3. Bojańczyk, M.: Transducers of polynomial growth. In: Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science. pp. 1–27 (2022)
4. Bojańczyk, M., Kiefer, S., Lhote, N.: String-to-string interpretations with polynomial-size output. In: 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019 (2019)
5. Chytil, M.P., Jákl, V.: Serial composition of 2-way finite-state transducers and simple programs on strings. In: 4th International Colloquium on Automata, Languages, and Programming, ICALP 1977. pp. 135–147. Springer (1977)
6. Colcombet, T.: Green's relations and their use in automata theory. In: International Conference on Language and Automata Theory and Applications. pp. 1–21. Springer (2011)
7. Dave, V., Gastin, P., Krishna, S.N.: Regular transducer expressions for regular transformations. In: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science. pp. 315–324. ACM (2018)
8. Douéneau-Tabot, G.: Pebble transducers with unary output. In: 46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021 (2021)
9. Douéneau-Tabot, G., Filiot, E., Gastin, P.: Register transducers are marble transducers. In: 45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020 (2020)
10. Douéneau-Tabot, G.: Hiding pebbles when the output alphabet is unary. In: 49th International Colloquium on Automata, Languages, and Programming, ICALP 2022 (2022)
11. Engelfriet, J.: Two-way pebble transducers for partial functions and their composition. Acta Informatica **52**(7-8), 559–571 (2015)

12. Engelfriet, J., Hoogeboom, H.J.: MSO definable string transductions and two-way finite-state transducers. ACM Transactions on Computational Logic (TOCL) **2**(2), 216–254 (2001)
13. Engelfriet, J., Hoogeboom, H.J., Samwel, B.: Xml transformation by tree-walking transducers with invisible pebbles. In: Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 63–72. ACM (2007)
14. Gurari, E.M.: The equivalence problem for deterministic two-way sequential transducers is decidable. SIAM Journal on Computing **11**(3), 448–452 (1982)
15. Hopcroft, J.E., Ullman, J.D.: An approach to a unified theory of automata. The Bell System Technical Journal **46**(8), 1793–1829 (1967)
16. Kiefer, S., Nguyên, L.T.D., Pradic, C.: Revisiting the growth of polyregular functions: output languages, weighted automata and unary inputs. arXiv preprint arXiv:2301.09234 (2023)
17. Nguyên, L.T.D., Noûs, C., Pradic, C.: Comparison-free polyregular functions. In: 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021 (2021)
18. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. IBM Journal of Research and Development **3**(2), 198–200 (1959)
19. Simon, I.: Factorization forests of finite height. Theor. Comput. Sci. **72**(1), 65–94 (1990)

Fixed Points and Noetherian Topologies

Aliaume Lopez[1,2(✉)]

[1] Université Paris Cité, CNRS, IRIF, F-75013, Paris, France
alopez@irif.fr
[2] Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France.

Abstract. Noetherian spaces are a generalisation of well-quasi-orderings to topologies, that can be used to prove termination of programs. They find applications in the verification of transition systems, some of which are better described using topology. The goal of this paper is to allow the systematic description of computations using inductively defined datatypes via Noetherian spaces. This is achieved through a fixed point theorem based on a topological minimal bad sequence argument.

Keywords: Noetherian spaces · topology · well-quasi-orderings · initial algebras · Kruskal's Theorem · Higman's Lemma.

1 Introduction

Let $(\mathcal{E}, \leq)$ be a set endowed with a quasi-order. A sequence $(x_n)_n \in \mathcal{E}^{\mathbb{N}}$ is *good* whenever there exists $i < j$ such that $x_i \leq x_j$. A quasi-ordered set $(\mathcal{E}, \leq)$ is a *well-quasi-ordered* — abbreviated as wqo — if every sequence is good. By calling a sequence *bad* whenever it is not good, well-quasi-orderings are equivalently defined as having no infinite bad sequences. This generalisation of well-founded total orderings can be used as a basis for proving program termination. For instance, algorithms alike Example 1.1 can be studied via well-quasi-orderings and the length of their bad sequences [5]. More generally, one can map the states of a run to a wqo via a so-called quasi-ranking function to both prove the termination of the program and gain information about its runtime [27, Chapter 2]. Let us provide a concrete example of this proof scheme.

Example 1.1. Let Alg be the algorithm with three integer variables a, b, c that non-deterministically performs one of the following operations until a, b or c becomes negative: (l) $\langle a, b, c\rangle \leftarrow \langle a-1, b, 2c\rangle$ or (r) $\langle a, b, c\rangle \leftarrow \langle 2c, b-1, 1\rangle$.

Lemma 1.2. *For every choice of $a, b, c \in \mathbb{N}^3$, the algorithm* Alg *terminates.*

Proof. Let us prove that Alg builds a bad sequence of triples when ordering $\mathbb{N}^3$ with $(a_1, b_1, c_1) \leq (a_2, b_2, c_2)$ whenever $a_1 \leq a_2$, $b_1 \leq b_2$, and $c_1 \leq c_2$. If (a_i, b_i, c_i) and (a_j, b_j, c_j) represent two configurations in a run of Alg, either only rule (l) was fired and $a_j < a_i$, or rule (r) was fired as least once, and $b_j < b_i$.

Because $(\mathbb{N}^3, \leq)$ is a well-quasi-ordering (see Dickson's Lemma in [28]), Alg terminates for every choice of initial triple $(a, b, c) \in \mathbb{N}^3$.

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 456–476, 2023.
https://doi.org/10.1007/978-3-031-30829-1_22

As a combinatorial tool, well-quasi-orderings appear frequently in varying fields of computer science, ranging from graph theory to number theory [18, 22, 21, 3]. Well-quasi-orderings have also been highly successful in proving the termination of verification algorithms. One critical application of well-quasi-orderings is to the verification of infinite state transition systems, via the study of so-called Well-Structured Transition Systems (WSTS) [1, 2, 16, 7].

Noetherian spaces. A major roadblock arises when using well-quasi-orders: the powerset of a well-quasi-order may fail to be one itself [26]. This is particularly problematic in the study of WSTS, where the powerset construction appears frequently [19, 29, 1]. To tackle this issue, one can justify that the quasi-orders of interest are not pathological, and are actually better quasi-orders [25, 23]. Another approach is offered by the topological notion of *Noetherian space*, which as pointed out by Goubault-Larrecq, can act as a suitable generalisation of well-quasi-orderings that is preserved under the powerset construction [10].

The topological analogues to WSTS enjoy similar decidability properties, and there even exists an analogue to Karp and Miller's forward analysis for Petri nets [11]. Moreover, their topological nature allows to verify systems beyond the reach of quasi-orderings, such as lossy concurrent polynomial programs [11]. This is possible because the polynomials are handled via results from algebraic geometry, through the notion of the *Zariski topology* over $\mathbb{C}^n$ [12, Exercise 9.7.53].

One drawback of the topological approach is that many topologies correspond to a single quasi-ordering. Hence, when the problem is better described via an ordering, one has to choose a specific topology, and there usually does not exist a finest one that is Noetherian.

Inductively defined datatypes. As for well-quasi-orders, Noetherian spaces are stable under finite products and finite sums [28, 12]. While this can be enough to describe the set of configurations of a Petri net using $\mathbb{N}^k$, it does not allow to talk about more complex data structures, that are typically defined inductively, such as lists and trees. To make the above statement precise, let $\mathbf{1}$ be the singleton set, $A + B$ be the disjoint union of A and B, and $A \times B$ their cartesian product. Then, the set of finite words over an alphabet Σ is precisely the least fixed point of $F\colon X \mapsto \mathbf{1} + \Sigma \times X$. Similarly, the set of finite trees over Σ equals $\mathsf{lfp}_X.\Sigma \times X^*$, where $\mathsf{lfp}_X.F(X)$ denotes the least fixed point of F.

In the realm of well-quasi-orderings, the specific cases of finite words and finite trees are handled respectively via Higman's Lemma [18] and Kruskal's Tree Theorem [22]. Let us recall that a word u *embeds* into a word w (written $u \leq_* v$) whenever whenever there exists a strictly increasing map $h\colon |w| \to |w'|$ such that $w_i \leq w_{h(i)}$ for $1 \leq i \leq |w|$. Similarly, a tree t *embeds* into a tree t' (written $t \leq_{\mathsf{tree}} t'$) whenever there exists a map from nodes of t to nodes of t' respecting the least common ancestor relation, and increasing the colours of the nodes. Proofs that finite words and finite trees preserve well-quasi-orderings typically rely on a so-called *minimal bad sequence argument* due to Nash-Williams [24]. However, the argument is quite subtle, and needs to be handled with care [9, 30].

In addition, the argument is not compositional and has to be slightly modified whenever a new inductive construction is desired [as in, e.g., 4, 3].

This picture has been adapted by Goubault-Larrecq to the topological setting by proposing analogues of the word embedding and tree embedding, together with a proof that they preserve Noetherian spaces [12, Section 9.7]. However, both the definitions and the proofs have an increased complexity, as they rely on an adapted "topological minimal bad sequence argument" that appears to be even more subtle [14, errata n. 26]. Moreover, the newly introduced topologies have involved definitions often relying on ad-hoc constructions.

In the case of well-quasi-orderings, two generic fixed point constructions have been proposed to handle inductively defined datatypes [17, 8]. In these frameworks, $\mathsf{lfp}_X.F(X)$ is guaranteed to be a well-quasi-ordering provided that F is a "well-behaved functor" of quasi-orders. Both proposals, while relying on different categorical notions, successfully recover Higman's word embedding and Kruskal's tree embedding through their respective definitions as least fixed points. As a side effect, they reinforce the idea that these two quasi-orders are somehow canonical.

In the case of Noetherian spaces, no equivalent framework exists to build inductive datatypes, and the notions of "well-behaved" constructors from [17, 8] rule out the use of important Noetherian spaces, as they require that an element $a \in F(X)$ has been built using *finitely many* elements of X: while this is the case for finite words and finite trees, it does not hold for the arbitrary powerset. Moreover, there have been recent advances in placing Noetherian topologies over spaces that are not straightforwardly obtained through "well-behaved" definitions, such as infinite words [13], or even ordinal length words [15].

1.1 Contributions of this paper

In this paper, we propose a least fixed point theorem for Noetherian topologies. This is done in a way that greatly differs from the categorical frameworks introduced in the study of well-quasi-orders, as the construction of the space is entirely *decoupled* from the construction of the topology. In particular, the carrier set X itself need not be inductively defined.

In this setting, we consider a fixed set X and a map R from topologies τ over X to topologies $R(\tau)$ over X. Because the set of topologies over X is a complete lattice, it suffices to ask for R to be monotone to guarantee that it has a least fixed point, that we write $\mathsf{lfp}_\tau.R(\tau)$. In general, this least fixed point will not be Noetherian, but we show that a simple sufficient condition on R guarantees that it is. This main theorem (Theorem 3.21), encapsulates all the complexity of the topological adaptations of the minimal bad sequences arguments [12, Section 9.7], and we believe that it has its own interest.

The necessity to separate the construction of the set of points from the construction of the topology might be perceived as a weakness of the theory, when it is in fact a strength of our approach. We illustrate this by giving a shorter proof that the words of ordinal length are Noetherian [15], without providing an

inductive definition of the space. As an illustration of the versatility of our framework, we introduce a reasonable topology over ordinal branching trees (with finite depth), and prove that it is Noetherian using the same technique.

In the specific cases where the space of interest can be obtained as a least fixed point of a "well-behaved" functor, we show how Theorem 3.21 can be used to generalise the categorical framework of Hasegawa [17] to a topological setting. As well as adding inductively defined topologies (hence, inductively defined datatypes) to the theory of Noetherian spaces, this provide a reasonable answer to the canonicity issue previously mentioned.

Outline. In Section 2 we recall some of the main results in the theory of Noetherian spaces. In Section 3 we prove our main result (Theorem 3.21). In Section 4 we explore how this result covers existing topological results in the literature, and provide a new non-trivial Noetherian space (Definition 4.7). In Section 5, we leverage our main result to devise a Noetherian topology over inductively defined datatypes (Theorem 5.13), and prove that this generalises the work of Hasegawa over well-quasi-orders (Theorem 5.15).

2 A Quick Primer on Noetherian Topologies

A *topological space* is a pair $(\mathcal{X}, \tau)$ where $\tau \subseteq \mathbb{P}(\mathcal{X})$, τ is stable under finite intersections, and τ is stable under arbitrary unions. A subset $U \subseteq \mathcal{X}$ is an *open subset* when $U \in \tau$, and a *closed subset* when $\mathcal{X} \setminus U \in \tau$. As an order-theoretic counterpart to open and closed subsets, we say that a subest U of a quasi-ordered set $(\mathcal{E}, \leq)$ is *upwards-closed* whenever for all $x \in U$, $x \leq y$ implies $y \in U$. Similarly, a subset is *downwards-closed* whenever its complement is upwards-closed. One can convert back and forth between the two as follows:

Notation 2.1. Let $(\mathcal{E}, \leq)$ be a quasi-order and $(\mathcal{X}, \tau)$ be a topological space. The *Alexandroff topology* $\mathsf{alex}(\leq)$ over $\mathcal{E}$ is the collection of upwards-closed subsets of $\mathcal{E}$. The *specialisation preorder* $\leq_\tau$ is defined via $x \leq_\tau y$ whenever for every open subset $U \in \tau$, if $x \in U$ then $y \in U$.

It is an easy check that the specialisation pre-order of the Alexandroff topology of a quasi-order $\leq$ is the quasi-order itself. Beware that several topologies can share the same specialisation pre-order $\leq$, and among those, the Alexandroff topology is the finest.

We can now build the topological analogue to wqos through the notion of compactness: a subset K of $\mathcal{X}$ is defined as *compact* whenever from every family $(U_i)_{i \in I}$ of open sets such that $K \subseteq \bigcup_{i \in I} U_i$, one can extract a finite subset $J \subseteq I$ such that $K \subseteq \bigcup_{i \in J} U_i$. A quasi-order $(\mathcal{E}, \leq)$ is wqo if and only if every subset K of $\mathcal{E}$ is compact for $\mathsf{alex}(\leq)$. Generalising this property to arbitrary topological spaces $(\mathcal{X}, \tau)$, a topological space $(\mathcal{X}, \tau)$ is said to be a *Noetherian space* whenever every subset of $\mathcal{X}$ is compact.

Table 1. An *algebra* of Noetherian spaces [see 10, 12, 15].

Constructor	Syntax	Topology
Well-quasi-orders	$\mathcal{E}$	Alexandroff topology
Complex vectors	$\mathbb{C}^k$	Zariski topology
Disjoint sum	$\mathcal{X}_1 + \mathcal{X}_2$	co-product topology
Product	$\mathcal{X}_1 \times \mathcal{X}_2$	product topology
Finite words	$\mathcal{X}^*$	subword topology
Finite trees	$\mathsf{T}(\mathcal{X})$	tree topology
Finite multisets	$\mathcal{X}^{\circledast}$	multiset topology
Transfinite words	$\mathcal{X}^{<\alpha}$	transfinite subword topology
Powerset	$\mathsf{P}(\mathcal{X})$	Lower-Vietoris

Remark 2.2. A space $(\mathcal{X}, \tau)$ is Noetherian if and only if for every increasing sequence of open subsets $(U_i)_{i\in\mathbb{N}}$, there exists $j \in \mathbb{N}$ such that $\bigcup_{i\in\mathbb{N}} U_i = \bigcup_{i\leq j} U_i$.

In order to inductively define Noetherian spaces, we will often rely on basic constructors such as the disjoint sum and the finite product. For completeness, we recall in Table 1 usual constructors that preserve Noetherian spaces. This table also illustrate the versatility of the concept, that encompasses both the algebraic properties of $\mathbb{C}^k$ and the order properties of well-quasi-orders.

3 Refinements of Noetherian topologies

Let us fix a set $\mathcal{X}$. The collection of topologies over $\mathcal{X}$ is itself a set, and forms a complete lattice for inclusion. In this lattice, the least element is the *trivial topology* $\tau_{\text{triv}} := \{\emptyset, \mathcal{X}\}$, and the greatest element is the *discrete topology* $\mathbb{P}(\mathcal{X})$. Thanks to Tarski's fixed point theorem, every monotone function R mapping topologies over $\mathcal{X}$ to topologies over $\mathcal{X}$ has a least fixed point, which can be obtained by transfinitely iterating R from the trivial topology. Writing $\mathsf{lfp}_\tau.R(\tau)$ for the least fixed point of R, our goal is to provide sufficient conditions for $(\mathcal{X}, \mathsf{lfp}_\tau.R(\tau))$ to be Noetherian.

Definition 3.1. *A* refinement function *over a set $\mathcal{X}$ is a function* R *mapping topologies over $\mathcal{X}$ to topologies over $\mathcal{X}$. Moreover, we assume that $\mathsf{R}(\tau)$ is Noetherian whenever τ is, and that $\mathsf{R}(\tau) \subseteq \mathsf{R}(\tau')$ when $\tau \subseteq \tau'$.*

As $(\mathcal{X}, \tau_{\text{triv}})$ is always Noetherian, $(\mathcal{X}, \mathsf{R}^n(\tau_{\text{triv}}))$ is Noetherian for all $n \in \mathbb{N}$ and refinement function R. However, it remains unclear whether the transfinite iterations needed to reach a fixed point preserve Noetherian spaces.

We demonstrate in Example 3.2 how to obtain the topology $\mathsf{alex}(\leq)$ over $\mathbb{N}$ as a least fixed point of some simple refinement function. Before that, let us define the notion of upwards-closure: given a quasi-order $(\mathcal{E}, \leq)$ and a set $E \subseteq \mathcal{E}$, let us define the *upwards-closure* of E, written $\uparrow_{\leq} E$, as the set of elements that are greater or equal than some element of E in $\mathcal{E}$.

Example 3.2 (Natural Numbers). Over $X := \mathbb{N}$, one can define $\mathsf{Div}(\tau)$ as the collection of the sets $\uparrow_{\leq} (U+1)$ for $U \in \tau$, plus $\mathbb{N}$ itself. Then $\mathsf{Div}(\tau_{\mathrm{triv}}) = \{\emptyset, \uparrow_{\leq} 1, \mathbb{N}\}$, $\mathsf{Div}^2(\tau_{\mathrm{triv}}) = \{\emptyset, \uparrow_{\leq} 1, \uparrow_{\leq} 2, \mathbb{N}\}$. More generally, for every $k \geq 0$, $\mathsf{Div}^k(\tau_{\mathrm{triv}}) = \{\emptyset, \uparrow_{\leq} 1, \ldots, \uparrow_{\leq} k, \mathbb{N}\}$. It is an easy check that $\mathsf{lfp}_{\tau}.\mathsf{Div}(\tau)$ is precisely $\mathsf{alex}(\leq)$, which is Noetherian because $(\mathbb{N}, \leq)$ is a well-quasi-ordering.

3.1 An ill-behaved refinement function

Not all refinement functions behave as nicely as in Example 3.2, and one can obtain non-Noetherian topologies via their least fixed points.

Let us consider for this section $\Sigma := \{a, b\}$ with the discrete topology, i.e., $\{\emptyset, \{a\}, \{b\}, \Sigma\}$. Let us now build the set Σ^* of finite words over Σ. Whenever U and V are subsets of Σ^*, let us write UV for their concatenation, defined as $\{uv \colon u \in U, v \in V\}$. To construct an ill-behaved refinement function, we will associate to a topology τ the set $\{UV \colon U \in \{\emptyset, \{a\}, \{b\}, \Sigma\}, V \in \tau\}$. However, the latter fails to be a topology in general. This problem frequently appears in this paper, and is solved by considering the so-called generated topology.

Let us briefly recall that for every set $\mathcal{X}$ and collection of subsets $B \subseteq \mathbb{P}(\mathcal{X})$, one can construct the topology generated from B as the least topology on $\mathcal{X}$ containing B. This topology coincides with the one containing arbitrary unions of finite intersections of subsets in B. We say that B is a *subbasis* of τ when τ is the topology generated by B. Alexanders's Subbase Lemma allows to study Noetherian spaces in this setting [12, Thm. 4.4.29]: it states that checking whether a subset K of $\mathcal{X}$ is compact in τ can be done by considering only open subsets in B, i.e., that for every family $(U_i)_{i \in I}$ of a subbasis B of τ such that $K \subseteq \bigcup_{i \in I} U_i$, one can extract a finite subset $J \subseteq I$ such that $K \subseteq \bigcup_{j \in J} U_j$.

Definition 3.3. *Let* $\mathsf{R}_{\mathrm{pref}}$ *be the function mapping a topology* τ *over* Σ^* *to the topology generated by the sets* UV *where* $U \subseteq \Sigma$ *and* $V \in \tau$,

We refer to Figure 1 for a graphical presentation of the first two iterations of the refinement function $\mathsf{R}_{\mathrm{pref}}$. For the sake of completeness, let us compute $\mathsf{lfp}_{\tau}.\mathsf{R}_{\mathrm{pref}}(\tau)$, which is the Alexandroff topology of the prefix ordering on words.

Definition 3.4. *The* prefix topology[3] τ_{pref^*}*, over* Σ^* *is generated by the following open sets:* $U_1 \ldots U_n \Sigma^*$*, where* $n \geq 0$ *and* $U_i \subseteq \Sigma$.

Lemma 3.5. *The prefix topology over* Σ^* *is the least fixed point of* $\mathsf{R}_{\mathrm{pref}}$.

Lemma 3.6. *The function* $\mathsf{R}_{\mathrm{pref}}$ *is a refinement function.*

Proof. It is an easy check that whenever $\tau \subseteq \tau'$, $\mathsf{R}_{\mathrm{pref}}(\tau) \subseteq \mathsf{R}_{\mathrm{pref}}(\tau')$. Now, assume that τ is Noetherian, it remains to prove that $\mathsf{R}_{\mathrm{pref}}(\tau)$ remains Noetherian. Consider a subset $E \subseteq \Sigma^*$ and let us prove that E is compact in $\mathsf{R}_{\mathrm{pref}}(\tau)$.

[3] This definition differs from what is called the "prefix topology" in the literature [see 6, 12, resp. Section 8 and Exercise 9.7.36].

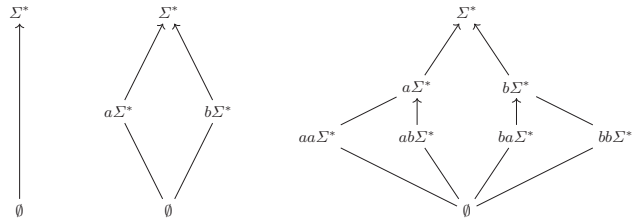


Fig. 1. Iterating $\mathsf{R}_{\mathrm{pref}}$ over Σ^*. On the left the trivial topology τ_{triv}, followed by $\mathsf{R}_{\mathrm{pref}}$, and on the right $\mathsf{R}_{\mathrm{pref}}{}^2$.

For that, we consider an open cover $E \subseteq \bigcup_{i\in I} W_i$, where $W_i \in \mathsf{R}_{\mathrm{pref}}(\tau)$. Thanks to Alexander's subbase lemma, we can assume without loss of generality that W_i is a subbasic open set of $\mathsf{R}_{\mathrm{pref}}(\tau)$, that is, $W_i = U_iV_i$ with $U_i \subseteq \Sigma$ and $V_i \in \tau$.

Since $(\Sigma^*, \tau) \times (\Sigma^*, \tau)$ is Noetherian (see Table 1), there exists a finite set $J \subseteq I$ such that $\bigcup_{i\in J} U_i \times V_i = \bigcup_{i\in I} U_i \times V_i$. This implies that $E \subseteq \bigcup_{i\in J} U_iV_i$, and provides a finite subcover of E. □

The sequence $\bigcup_{0\leq i\leq k} a^ib\Sigma^*$, for $k \in \mathbb{N}$, is a strictly increasing sequence of opens. Therefore, the prefix topology is not Noetherian. The terms $a^ib\Sigma^*$ can be observed in Figure 1 as a diagonal of incomparable open sets.

Corollary 3.7. *The topology* $\mathsf{lfp}_\tau.\mathsf{R}_{\mathrm{pref}}(\tau)$ *is not Noetherian.*

The prefix topology is not Noetherian, even when starting from a finite alphabet. However, we claimed in Section 1 that there is a natural generalisation of the subword embedding to topological spaces which is Noetherian. Before introducing this topology, let us write $[U_1, \ldots, U_n]$ as a shorthand notation for the set $\Sigma^*U_1\Sigma^* \ldots \Sigma^*U_n\Sigma^*$.

Definition 3.8 (Subword topology [12, Definition 9.7.26]). *Given a topological space* (Σ, τ)*, the space* Σ^* *of finite words over* Σ *can be endowed with the* subword topology*, generated by the open sets* $[U_1, \ldots, U_n]$ *when* $U_i \in \tau$*.*

The *topological Higman lemma* [12, Theorem 9.7.33] states that the subword topology over Σ^* is Noetherian if and only if Σ is Noetherian. Although the subword topology might seem ad-hoc, it can be validated as a generalisation of the subword embedding because the subword topology of $\mathsf{alex}(\leq)$ equals the Alexandroff topology of the subword ordering of $\leq$, for every quasi-order $\leq$ over Σ [12, Exercise 9.7.30]. Let us now reverse engineer a refinement function whose least fixed point is the subword topology.

Definition 3.9. *Let* (Σ, θ) *be a topological space. Let* $\mathsf{E}^\theta_{\mathrm{words}}$ *be defined as mapping a topology* τ *over* Σ^* *to the topology generated by the following sets:* $\uparrow_{\leq_*} UV$ *for* $U, V \in \tau$*; and* $\uparrow_{\leq_*} W$*, for* $W \in \theta$*.*

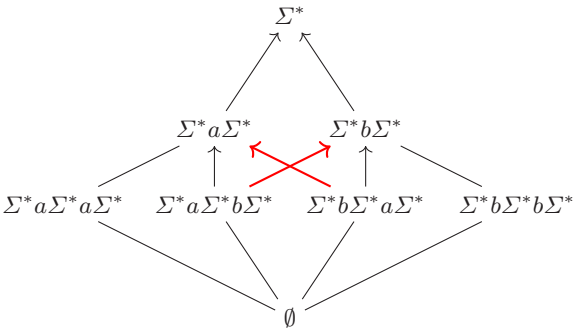


Fig. 2. The topology $\mathsf{E}^{\theta}_{\text{words}}{}^{2}(\tau_{\text{triv}})$, with bold red arrows for the inclusions that were not present between the "analogous sets" in $\mathsf{R}_{\text{pref}}{}^{2}(\tau_{\text{triv}})$. We have taken θ to be the discrete topology over Σ.

Lemma 3.10. *Let (Σ, θ) be a topological space. The subword topology over Σ^* is the least fixed point of $\mathsf{E}^{\theta}_{\text{words}}$.*

In order to show that $\mathsf{E}^{\theta}_{\text{words}}$ is a refinement function, we first claim that the two parts of the topology can be dealt with separately.

Lemma 3.11 ([12, Proposition 9.7.18]). *If $(\mathcal{X}, \tau)$ and $(\mathcal{X}, \tau')$ are Noetherian, then $\mathcal{X}$ endowed with the topology generated by $\tau \cup \tau'$ is Noetherian.*

Lemma 3.12. *Let (Σ, θ) be a Noetherian topological space. The map $\mathsf{E}^{\theta}_{\text{words}}$ is a refinement function over Σ.*

Proof. We leave the monotonicity of $\mathsf{E}^{\theta}_{\text{words}}$ as an exercice and focus on the proof that $\mathsf{E}^{\theta}_{\text{words}}(\tau)$ is Noetherian, whenever τ is. Thanks to Lemma 3.11, it suffices to prove that the topology generated by the sets $\uparrow_{\leq_*} UV$ (U, V open in τ), and the topology generated by the sets $\uparrow_{\leq_*} W$ (W open in θ) are Noetherian.

Let $(\uparrow_{\leq_*} U_i V_i)_{i\in\mathbb{N}}$ be a sequence of open sets. Because Noetherian topologies are closed under products (see Table 1), there exists k such that $\bigcup_{i\leq k} U_i \times V_i = \bigcup_{i\in\mathbb{N}} U_i \times V_i$. Hence, $\bigcup_{i\leq k} \uparrow_{\leq_*} U_i V_i = \bigcup_{i\in\mathbb{N}} \uparrow_{\leq_*} U_i V_i$

Let $\uparrow_{\leq_*} W_i$ be a sequence of open sets. Because θ is Noetherian, there exists k such that $\bigcup_{i\leq k} W_i = \bigcup_{i\in\mathbb{N}} W_i$, hence $\bigcup_{i\leq k} \uparrow_{\leq_*} W_i = \bigcup_{i\in\mathbb{N}} \uparrow_{\leq_*} W_i$. □

We have designed two refinement functions R_{pref} and $\mathsf{E}^{\theta}_{\text{words}}$ over Σ^*. Fixing θ to be the discrete topology over Σ, the least fixed point of R_{pref} is not Noetherian while the least fixed point of $\mathsf{E}^{\theta}_{\text{words}}$ is. We have depicted the result of iterating $\mathsf{E}^{\theta}_{\text{words}}$ twice over the trivial topology in Figure 2. As opposed to R_{pref}, the "diagonal" elements are comparable for inclusion.

3.2 Well-behaved refinement functions

In this section, we will show how the behaviour of refinement function with respect to subsets will act as a sufficient condition to separate the well-behaved

ones from the others. In order to make the idea of computing the refinement function directly over a subset precise, we will replace a subset with the induced topology by a "restricted" topology over the whole space.

Definition 3.13. *Let $(\mathcal{X}, \tau)$ be a topological space and H be a closed subset of $\mathcal{X}$. Define the* subset restriction *$\tau|H$ to be the topology generated by the open subsets $U \cap H$ where U ranges over τ.*

Let $\mathcal{X}$ be a topological space, and H be a proper closed subset of $\mathcal{X}$. The space $\mathcal{X}$ endowed with $\tau|H$ has a lattice of open sets that is isomorphic to the one of the space H endowed with the topology induced by τ, except for the entire space $\mathcal{X}$ itself. As witnessed by Example 3.14, the two spaces are in general not homeomorphic.

Example 3.14. Let $\mathbb{R}$ be endowed with the usual metric topology. The set $\{a\}$ is a closed set when $a \in \mathbb{R}$. The induced topology over $\{a\}$ is $\{\emptyset, \{a\}\}$. The subset restriction of the topology to $\{a\}$ is $\tau_a := \{\emptyset, \{a\}, \mathbb{R}\}$. Clearly, $(\mathbb{R}, \tau_a)$ and $(\{a\}, \tau_{\text{triv}})$ are not homeomorphic.

In order to build intuition, let us consider the special case of an Alexandroff topology over $\mathcal{X}$ and compute the specialisation preorder of $\tau|H$, where H is a downwards closed set.

Lemma 3.15. *Let $\tau = \mathsf{alex}(\leq)$ over a set $\mathcal{X}$, and $x, y \in X$. Then, $x \leq_{\tau|H} y$ if and only if $x \leq_\tau y \wedge y \in H$ or $x \notin H$. In other words, H^c is collapsed to an equivalence class below H itself.*

Definition 3.16. *A* topology expander *is a refinement function E that satisfies the following extra property: for every Noetherian topology τ satisfying $\tau \subseteq \mathsf{E}(\tau)$, for all closed set H in τ, $\mathsf{E}(\tau)|H \subseteq \mathsf{E}(\tau|H)|H$.*

Lemma 3.17. *The refinement function R_{pref} is not a topology expander.*

Proof. Let us consider $\tau := \{\emptyset, a\Sigma^*, b\Sigma^*, \Sigma^*\}$. Remark that $H := a\Sigma^* \cup \{\varepsilon\}$ is a closed subset because $\Sigma = \{a, b\}$. It is an easy check that $\mathsf{R}_{\text{pref}}(\tau)|H = \{\emptyset, aa\Sigma^*, ab\Sigma^*, a\Sigma^*, \Sigma^*\} \neq \{\emptyset, aa\Sigma^*, a\Sigma^*, \Sigma^*\} = \mathsf{R}_{\text{pref}}(\tau|H)|H$.

Lemma 3.18. *When θ is Noetherian, $\mathsf{E}^\theta_{\text{words}}$ is a topology expander.*

Proof. We have proven in Lemma 3.12 that $\mathsf{E}^\theta_{\text{words}}$ is a refinement function. Let us now prove that it is a topology expander.

Let τ be a Noetherian topology over Σ^*, such that $\tau \subseteq \mathsf{E}^\theta_{\text{words}}(\tau)$. Let H be a closed subset of (Σ^*, τ). Notice that as H is closed in τ, and since $\tau \subseteq \mathsf{E}^\theta_{\text{words}}(\tau)$, H is downwards closed for $\leq_*$. As a consequence, $(\uparrow_{\leq_*} UV) \cap H = (\uparrow_{\leq_*} (U \cap H)(V \cap H)) \cap H$. Hence, $\mathsf{E}^\theta_{\text{words}}(\tau)|H \subseteq \mathsf{E}^\theta_{\text{words}}(\tau|H)|H$. □

3.3 Iterating Expanders

Our goal is now to prove that topology expanders are refinement functions that can be safely iterated. For that, let us first define precisely what "iterating transfinitely" a refinement function means.

Definition 3.19. *Let $(\mathcal{X}, \tau)$ be a topological space, and E be a topology expander. The* limit topology $\mathsf{E}^\alpha(\tau)$ *is defined as: τ when $\alpha = 0$, $\mathsf{E}(\mathsf{E}^\beta(\tau))$ when $\alpha = \beta + 1$, and as the join of the topologies $\mathsf{E}^\beta(\tau)$ for all $\beta < \alpha$, when α is a limit ordinal.*

We devote the rest of this section to proving our main theorem, which immediately implies that least fixed points of topology expanders are Noetherian. Notice that the theorem is trivial whenever α is a successor ordinal.

Proposition 3.20. *Let α be an ordinal, τ be a topology, and E be a topology expander. If $\mathsf{E}^\beta(\tau)$ is Noetherian for all $\beta < \alpha$, and $\tau \subseteq \mathsf{E}(\tau)$, then $\mathsf{E}^\alpha(\tau)$ is Noetherian.*

Theorem 3.21 (Main Result). *Let $\mathcal{X}$ be a set and E be a topology expander. The least fixed point of E is a Noetherian topology over $\mathcal{X}$.*

The topological minimal bad sequence argument. In order to prove Theorem 3.21, we will use a topological minimal bad sequence argument. To that end, let us first introduce a well-founded partial ordering over the elements of $\mathsf{E}^\alpha(\tau)$. With an open set $U \in \mathsf{E}^\alpha(\tau)$, we associate a depth $\mathsf{depth}(U)$, defined as the smallest ordinal $\beta \leq \alpha$ such that $U \in \mathsf{E}^\beta(\tau)$. We then define $U \trianglelefteq V$ to hold whenever $\mathsf{depth}(U) \leq \mathsf{depth}(V)$, and $U \triangleleft V$ whenever $\mathsf{depth}(U) < \mathsf{depth}(V)$. It is an easy check that this is a well-founded total quasi-order over $\mathsf{E}^\alpha(\tau)$.

As a first step towards proving that $\mathsf{E}^\alpha(\tau)$ is Noetherian for a limit ordinal α, we first reduce the problem to open subsets of depth strictly less than α itself.

Lemma 3.22. *Let α be a limit ordinal, and E be a topology expander. The topology $\mathsf{E}^\alpha(\tau)$ has a subbasis of elements of depth strictly below α.*

Let us recall the notion of topological bad sequence designed by Goubault-Larrecq [12, Lemma 9.7.31] in the proof of the Topological Kruskal Theorem, adapted to our ordering of subbasic open sets.

Definition 3.23. *Let $(\mathcal{X}, \tau)$ be a topological space. A sequence $\mathcal{U} = (U_i)_{i \in \mathbb{N}}$ of open subsets is* good *if there exists $i \in \mathbb{N}$ such that $U_i \subseteq \bigcup_{j<i} U_j$. A sequence that is not good is called* bad.

Lemma 3.24. *Let α be a limit ordinal, and E be a topology expander such that $\mathsf{E}^\alpha(\tau)$ is not Noetherian. Then, there exists a bad sequence $\mathcal{U}$ of open subsets in $\mathsf{E}^\alpha(\tau)$ of depth less than α that is lexicographically minimal for $\trianglelefteq$. Such a sequence is called* minimal bad.

We deduce that in a limit topology, minimal bad sequences are not allowed to use open subsets of arbitrary depth. This will then be leveraged via Lemma 3.27 to decrease the depth by one.

Lemma 3.25. *Let α be a limit ordinal, τ be a topology and E be a topology expander such that $\mathsf{E}^\beta(\tau)$ is Noetherian for all $\beta < \alpha$. Assume that $\mathcal{U} = (U_i)_{i\in\mathbb{N}}$ is a minimal bad sequence of $\mathsf{E}^\alpha(\tau)$. Then, for every $i \in \mathbb{N}$, $\mathsf{depth}(U_i)$ is either 0 or a successor ordinal.*

Definition 3.26. *Let α be an ordinal, τ be a topology, E be a topology expander such that $\tau \subseteq \mathsf{E}(\tau)$, and let $U \in \mathsf{E}^\alpha(\tau)$. The topology $\mathsf{Down}(U)$ is generated by the open sets V such that $V \triangleleft U$, where V ranges over $\mathsf{E}^\alpha(\tau)$.*

Lemma 3.27. *Let α be an ordinal, E be a topology expander and $U \in \mathsf{E}^\alpha(\tau)$. If $\mathsf{depth}(U)$ is a successor ordinal, then $U \in \mathsf{E}(\mathsf{Down}(U))$.*

If $\mathcal{U}$ is a minimal bad sequence in $(X, \mathsf{E}^\alpha(\tau))$, then $U_i \not\subseteq \bigcup_{j<i} U_j := V_i$, i.e., $U_i \cap V_i^c \neq \emptyset$. We can now use our subset restriction operator to devise a topology associated to this minimal bad sequence. Noticing that $H_i := V_i^c$ is a closed set in $\mathsf{E}^\alpha(\tau)$, hence we can build the subset restriction $\mathsf{Down}(U_i)|H_i$.

Definition 3.28. *Let α be an ordinal, τ be a topology, E be a topology expander such that $\tau \subseteq \mathsf{E}(\tau)$, and let $\mathcal{U} = (U_i)_{i\in\mathbb{N}}$ be a minimal bad sequence in $\mathsf{E}^\alpha(\tau)$. Then, the* minimal topology *$\mathcal{U}(\mathsf{E}^\alpha(\tau))$ is generated by $\bigcup_{i\in\mathbb{N}} \mathsf{Down}(U_i)|H_i$, where $H_i := (\bigcup_{j<i} U_j)^c$.*

Lemma 3.29. *Let α be an ordinal, τ be a topology, E be a topology expander such that $\tau \subseteq \mathsf{E}(\tau)$, and let $\mathcal{U} = (U_i)_{i\in\mathbb{N}}$ be a minimal bad sequence in $\mathsf{E}^\alpha(\tau)$. Then, the minimal topology $\mathcal{U}(\mathsf{E}^\alpha(\tau))$ is Noetherian.*

Proof. Assume by contradiction that $\mathcal{U}(\mathsf{E}^\alpha(\tau))$ is not Noetherian. Let us define V_i as $\bigcup_{j<i} U_j$, and H_i as V_i^c.

Thanks to [12, Lemma 9.7.15] there exists a bad sequence $\mathcal{W} := (W_i)_{i\in\mathbb{N}}$ of subbasic elements of $\mathcal{U}(\mathsf{E}^\alpha(\tau))$. By definition, W_i is in some $\mathsf{Down}(U_j)|H_j$. Let us select a mapping $\rho\colon \mathbb{N} \to \mathbb{N}$, such that $W_i \in \mathsf{Down}(U_{\rho(i)})|H_{\rho(i)}$. This amounts to the existence of an open $T_{\rho(i)}$, such that $T_{\rho(i)} \triangleleft U_{\rho(i)}$, and $W_i = T_{\rho(i)} \setminus V_{\rho(i)}$. Without loss of generality we assume that ρ is monotonic.

Let us build the sequence $\mathcal{Y}$ defined by $Y_i := U_i$ if $i < \rho(0)$ and $Y_i := T_{\rho(i)}$ otherwise. This is a sequence of open sets in $\mathsf{E}^\alpha(\tau)$ that is lexicographically smaller than $\mathcal{U}$, hence $\mathcal{Y}$ is a good sequence: there exists $i \in \mathbb{N}$ such that $Y_i \subseteq \bigcup_{j<i} Y_j$.

- If $i < \rho(0)$, then $U_i \subseteq \bigcup_{j<i} U_j$ contradicting that $\mathcal{U}$ is bad.
- If $i \geq \rho(0)$, let us write $Y_i = T_{\rho(i)} \subseteq \bigcup_{j<\rho(0)} U_j \cup \bigcup_{j<i} T_{\rho(j)}$. By taking the intersection with $H_{\rho(i)}$, we obtain $W_i \subseteq \bigcup_{j<i} W_j$, contradicting the fact that $\mathcal{W}$ is a bad sequence. □

We are now ready to leverage our knowledge of minimal topologies associated with minimal bad sequences to carry on the proof of our main theorem.

Proposition 3.20. *Let α be an ordinal, τ be a topology, and E be a topology expander. If $\mathsf{E}^\beta(\tau)$ is Noetherian for all $\beta < \alpha$, and $\tau \subseteq \mathsf{E}(\tau)$, then $\mathsf{E}^\alpha(\tau)$ is Noetherian.*

Proof. If α is a successor ordinal, then $\alpha = \beta + 1$ and $\mathsf{E}^\alpha(\tau) = \mathsf{E}(\mathsf{E}^\beta(\tau))$. Because E respects Noetherian topologies, we immediately conclude that $\mathsf{E}^\alpha(\tau)$ is Noetherian. We are therefore only interested in the case where α is a limit ordinal.

Assume by contradiction that $\mathsf{E}^\alpha(\tau)$ is not Noetherian, using Lemma 3.24 there exists a minimal bad sequence $\mathcal{U} := (U_i)_{i\in\mathbb{N}}$. Let us write $d_i := \mathsf{depth}(U_i) < \alpha$. Thanks to Lemma 3.25, d_i is either 0 or a successor ordinal.

Because $\mathsf{E}^\beta(\tau)$ is Noetherian for $\beta < \alpha$, there are finitely many open subsets U_i at depth β for every ordinal $\beta < \alpha$. Indeed, if they were infinitely many, one would extract an infinite bad sequence of opens in $\mathsf{E}^\beta(\tau)$, which is absurd.

Furthermore, the sequence $(d_i)_{i\in\mathbb{N}}$ must be monotonic, otherwise $\mathcal{U}$ would not be lexicographically minimal. We can therefore construct a strictly increasing map $\rho\colon \mathbb{N} \to \mathbb{N}$ such that $0 < \mathsf{depth}(U_{\rho(j)})$ and $\mathsf{depth}(U_i) < \mathsf{depth}(U_{\rho(j)})$ whenever $0 \leq i < \rho(j)$.

Let us consider some $i = \rho(n)$ for some $n \in \mathbb{N}$. Let us write $V_i := \bigcup_{j<i} U_j$, and $H_i := X \setminus V_i$. The set V_i is open in $\mathsf{Down}(U_i)$ by construction of ρ, hence H_i is closed in $\mathsf{Down}(U_i)$. As E is a topology expander, we derive the following inclusions:

$$\begin{aligned}\mathsf{E}(\mathsf{Down}(U_i))|H_i &\subseteq \mathsf{E}(\mathsf{Down}(U_i)|H_i)|H_i \\ &\subseteq \mathsf{E}(\mathcal{U}(\mathsf{E}^\alpha(\tau)))|H_i\end{aligned}$$

Recall that $U_i \in \mathsf{E}(\mathsf{Down}(U_i))$ thanks to Lemma 3.27. As a consequence, $U_i \setminus V_i = W_i \setminus V_i$ for some open set W_i in $\mathsf{E}(\mathcal{U}(\mathsf{E}^\alpha(\tau)))$. Thanks to Lemma 3.29, and preservation of Noetherian topologies through topology expanders, the latter is a Noetherian topology. Therefore, $(W_{\rho(i)})_{i\in\mathbb{N}}$ is a good sequence. This provides an $i \in \mathbb{N}$ such that $W_{\rho(i)} \subseteq \bigcup_{\rho(j)<\rho(i)} W_{\rho(j)}$. In particular,

$$\begin{aligned}U_{\rho(i)} \setminus V_{\rho(i)} = W_{\rho(i)} \setminus V_{\rho(i)} &\subseteq \bigcup_{\rho(j)<\rho(i)} W_{\rho(j)} \setminus V_{\rho(i)} \subseteq \bigcup_{\rho(j)<\rho(i)} W_{\rho(j)} \setminus V_{\rho(j)} \\ &\subseteq \bigcup_{\rho(j)<\rho(i)} U_{\rho(j)} \setminus V_{\rho(j)} \subseteq \bigcup_{j<\rho(i)} U_j = V_{\rho(i)}\end{aligned}$$

This proves that $U_{\rho(i)} \subseteq V_{\rho(i)}$, i.e. that $U_{\rho(i)} \subseteq \bigcup_{j<\rho(i)} U_j$. Finally, this contradicts the fact that $\mathcal{U}$ is bad. □

We have effectively proven that being well-behaved with respect to closed subspaces is enough to consider least fixed points of refinement functions. This behaviour should become clearer in the upcoming sections, where we illustrate how this property can be ensured both in the case of Noetherian spaces and well-quasi-orderings.

4 Applications of Topology Expanders

We now briefly explore topologies that can be proven to be Noetherian using Theorem 3.21. It should not be surprising that both the topological Higman lemma and the topological Kruskal theorem fit in the framework of topology expanders, as both were already proven using a minimal bad sequence argument. However, we will proceed to extend the use of topology expander to spaces for which the original proof did not use a minimal bad sequence argument, and illustrate how they can easily be used to define new Noetherian topologies.

Finite words and finite trees. As a first example, we can easily recover the *topological Higman lemma* [12, Theorem 9.7.33] because the subword topology is the least fixed point of $\mathsf{E}^{\theta}_{\text{words}}$, which is a topology expander (see Lemmas 3.10 and 3.18).

It does not require much effort to generalise this proof scheme to the case of the *topological Kruskal theorem* [12, Theorem 9.7.46]. As a shorthand notation, let us write $t \in \diamond U\langle V\rangle$ whenever there exists a subtree t' of t whose root is labelled by an element of U and whose list of children belongs to V. Recall that we write $u \leq_* v$ when u is a scattered subword of v, and $t \leq_{\mathsf{tree}} t'$ when t embeds in t' as a tree (see page 2). As for the subword topology, the definition is ad-hoc but correctly generalises the tree embedding relation because the tree topology of $\mathsf{alex}(\leq)$ is the Alexandroff topology of $\leq_{\mathsf{tree}}$, for every ordering $\leq$ over Σ [12, Exercise 9.7.48].

Definition 4.1 ([12, Definition 9.7.39]). *Let (Σ, θ) be a topological space. The space $\mathsf{T}(\Sigma)$ of finite trees over Σ can be endowed with the* tree topology*, the coarsest topology such that $\diamond U\langle V\rangle$ is open whenever U is an open set of Σ, and V is an open set of $\mathsf{T}(\Sigma)^*$ in its subword topology.*

Definition 4.2. *Let (Σ, θ) be a topological space. Let $\mathsf{E}_{\text{tree}}{}^{\theta}$ be the function that maps a topology τ to the topology generated by the sets $\uparrow_{\leq_{\mathsf{tree}}} U\langle V\rangle$, for U open in θ, V open in $\mathsf{T}(\Sigma)^*$ with the subword topology of τ.*

Lemma 4.3. *The tree topology is the least fixed point of $\mathsf{E}_{\text{tree}}{}^{\theta}$, which is a topology expander. Hence, the tree topology is Noetherian when θ is.*

Ordinal words. Let us now demonstrate how Theorem 3.21 can be applied over spaces which are proved to be Noetherian without using a minimal bad sequence argument. For that, let us consider $\Sigma^{<\alpha}$ the set of words of ordinal length less than α, where α is a fixed ordinal. Since $\leq_*$ is in general not a wqo on $\Sigma^{<\alpha}$ when $\leq$ is wqo on Σ, this also provides an example of a topological minimal bad sequence argument that has no counterpart in the realm of wqos.

Definition 4.4 ([15]). *Let (Σ, θ) be a topological space. The* ordinal subword topology *over $\Sigma^{<\alpha}$ is the topology generated by the closed sets $F_1^{<\beta_1} \cdots F_n^{<\beta_n}$, for $n \in \mathbb{N}$, F_i closed in θ, $\beta_i < \alpha$, and where $F^{<\beta}$ is the set of words of length less than β with all of their letters in F.*

The ordinal subword topology is Noetherian [15], but the proof is quite technical and relies on the in-depth study of the possible inclusions between the subbasic closed sets. Before defining a suitable topology expander, given an ordinal β and a set $U \subseteq \Sigma^{<\alpha}$, let us write $w \in \beta \triangleright U$ if and only if $w_{>\gamma} \in U$ for all $0 \leq \gamma < \beta$.

Definition 4.5. *Let (Σ, θ) be a topological space, and α be an ordinal. The function $\mathsf{E}^{\theta}_{\alpha\text{-words}}$ maps a topology τ to the topology generated by the following sets: $\uparrow_{\leq_*} UV$ for U, V opens in τ; $\uparrow_{\leq_*} \beta \triangleright U$, for U open in τ, $\beta \leq \alpha$; $\uparrow_{\leq_*} W$, for W open in θ.*

Lemma 4.6. *Given a Noetherian space (Σ, θ), and an ordinal α. The map $\mathsf{E}^{\theta}_{\alpha\text{-words}}$ is a topology expander, whose least fixed point contains the ordinal subword topology. Therefore, the ordinal subword topology is Noetherian.*

Remark that Definitions 4.2, 4.5 and 3.9 all follow the same blueprint: new open sets are built as upwards closure for the corresponding quasi-order of the natural constructors associated to the space. We argue that this blueprint mitigates the canonicity issue and the complexity of Definitions 4.1, 4.4 and 3.8.

Ordinal branching trees. As an example of a new Noetherian topology derived using Theorem 3.21, we will consider *α-branching trees* $\mathsf{T}^{<\alpha}(\Sigma)$, i.e., the least fixed point of the constructor $X \mapsto \mathbf{1} + \Sigma \times X^{<\alpha}$ where α is a given ordinal. This example was not known to be Noetherian, and fails to be a well-quasi-order, and illustrates how Theorem 3.21 easily applies on inductively defined spaces.

Definition 4.7. *Let (Σ, θ) be a Noetherian space. The* ordinal tree topology *over α-branching trees is the least fixed point of $\mathsf{E}^{\theta}_{\alpha\text{-trees}}$, mapping a topology τ to the topology generated by the sets $\uparrow_{\leq_{\mathsf{tree}}} U\langle V \rangle$, where $U \in \theta$, V is open in $(\mathsf{T}^{<\alpha}(\Sigma))^{<\alpha}$ with the ordinal subword topology, and $U\langle V \rangle$ is the set of trees whose root is labelled by an element of U and list of children belongs to V.*

Theorem 4.8. *The α-branching trees endowed with the ordinal tree topology forms a Noetherian space.*

Proof. It suffices to prove that $\mathsf{E}^{\theta}_{\alpha\text{-trees}}$ is a topology expander. It is clear that $\mathsf{E}^{\theta}_{\alpha\text{-trees}}$ is monotone, and a closed set of $\mathsf{E}^{\theta}_{\alpha\text{-trees}}(\tau)$ is always downwards closed for $\leq_{\mathsf{tree}}$. As a consequence, if $\tau \subseteq \mathsf{E}^{\theta}_{\alpha\text{-trees}}(\tau)$ and H is closed in τ, $t \in V := (\uparrow_{\leq_{\mathsf{tree}}} U\langle V \rangle) \cap H$ if and only if $t \in H$ and every children of t belongs to H. Therefore, $(\uparrow_{\leq_{\mathsf{tree}}} U\langle V \rangle) \cap H = (\uparrow_{\leq_{\mathsf{tree}}} U\langle V \cap H^{<\alpha} \rangle) \cap H$. Notice that $H^{<\alpha} \cap V$ is an open of the ordinal subword topology over $\tau|H$. As a consequence, $V \cap H \in \mathsf{E}^{\theta}_{\alpha\text{-trees}}(\tau|H)|H$.

Let us now check that $\mathsf{E}^{\theta}_{\alpha\text{-trees}}$ preserves Noetherian topologies. Let $W_i := \uparrow_{\leq_{\mathsf{tree}}} U_i\langle V_i \rangle$ be a $\mathbb{N}$-indexed sequence of open sets in $\mathsf{E}^{\theta}_{\alpha\text{-trees}}(\tau)$ where τ is Noetherian. The product of the topology θ and the ordinal subword topology over τ is Noetherian thanks to Table 1 and Lemma 4.6. Hence, there exists a $i \in \mathbb{N}$ such that $U_i \times V_i \subseteq \bigcup_{j<i} U_j \times V_j$. As a consequence, $W_i \subseteq \bigcup_{j<i} W_j$. We have proven that $\mathsf{E}^{\theta}_{\alpha\text{-trees}}(\tau)$ is Noetherian. □

At this point, we have proven that the framework of topology expanders allows to build non-trivial Noetherian spaces. We argue that this bears several advantages over ad-hoc proofs: (i) the ad-hoc proofs are often tedious and error prone [12, 13, 15] (ii) the verification that E is a topology expander on the other hand is quite simple (iii) reduces the canonicity issue of topologies to the choice of a suitable topology expander.

5 Consequences on inductive definitions

So far, the process of constructing Noetherian spaces has been the following: first build a set of points, then compute a topology that is Noetherian as a least fixed point. In the case where the set of points itself is inductively defined (such as finite words or finite trees), the second step might seem redundant, and getting rid of it provides a satisfactory answer to the canonicity concerns about Noetherian topologies.

Before studying inductive definition of topological spaces, the notion of least fixed-point in this setting has to be made precise. To that purpose, let us now introduce ome basic notions of category theory. In this paper only three categories will appear, the category Set of sets and functions, the category Top of topological spaces and continuous maps, and the category Ord of quasi-ordered spaces and monotone maps. Using this language, a unary constructor G in the algebra of wqos defines an *endofunctor* from objects of the category Ord to objects of the category Ord preserving well-quasi-orderings.

Notation 5.1. Recall that in a category $\mathcal{C}$, $\mathrm{Hom}(A, B)$ is used to denote the collection of morphisms from the object A to the object B in $\mathcal{C}$. Moreover, $\mathrm{Aut}(A)$ denotes the set of *automorphisms* of A, i.e., invertible elements of $(\mathrm{Hom}(A, A), \circ)$.

In our study of Noetherian spaces (resp. well-quasi-orderings), we will often see constructors G' as first building a new set of structures, and then adapting the topology (resp. ordering) to this new set. In categorical terms, we are interested in endofunctors G' that are U-lifts of endofunctors on Set, where U is the forgetful functor from Top (resp. Ord) to Set.

5.1 Divisibility Topologies of Analytic Functors

The goal of this section is to introduce the categorical framework needed to formalise the automatic definition of a topology over an inductively defined datatype, and to compare this definition with the work that exists on well-quasi-orders by Hasegawa [17] and Freund [8]. We will avoid as much as possible the use of complex machinery related to analytic functors, and use as a definition an equivalent characterisation given by Hasegawa [17, Theorem 1.6]. For an introduction to analytic functors and combinatorial species, we redirect the reader to Joyal [20].

Notation 5.2. Given G an endofunctor of Set, the *category of elements* $\mathrm{el}(\mathsf{G})$ has as objects pairs (E, a) with $a \in \mathsf{G}(E)$, and as morphisms between (E, a) and (E', a') maps $f : E \to E'$ such that $\mathsf{G}_f(a) = a'$.

As an intuition to the unfamiliar reader, an element (E, a) in $\mathrm{el}(\mathsf{G})$ is a witness that a can be produced through G by using elements of E. Morphisms of elements are witnessing how relations between elements of $\mathsf{G}(E)$ and $\mathsf{G}(E')$ arise from relations between E and E'. As a way to define a "smallest" set of elements E such that a can be found in $\mathsf{G}(E)$, we rely on transitive objects. We recall that in a category $\mathcal{C}$, if X, A are two objects, the action of $\mathrm{Aut}(X)$ on $\mathrm{Hom}(X, A)$ is transitive when for every pair $f, g \in \mathrm{Hom}(X, A)$, there exists a $h \in \mathrm{Aut}(X)$ such that $f \circ h = g$.

Notation 5.3. A *transitive object* in a category $\mathcal{C}$ is an object X satisfying the following two conditions for every object A of $\mathcal{C}$: (a) the set $\mathrm{Hom}(X, A)$ in $\mathcal{C}$ is non-empty; (b) the right action of $\mathrm{Aut}(X)$ on $\mathrm{Hom}(X, A)$ by composition is transitive.

Notation 5.4. Given an object A in a category $\mathcal{C}$, one can build the *slice category* $\mathcal{C}/A$ whose objects are elements of $\mathrm{Hom}(B, A)$ when B ranges over objects of $\mathcal{C}$ and morphisms between $c_1 \in \mathrm{Hom}(B_1, A)$ and $c_2 \in \mathrm{Hom}(B_2, A)$ are maps $f : B_1 \to B_2$ such that $c_2 \circ f = c_1$.

This notion of slice category can be combined with the one of transitive object to build so-called "weak normal forms".

Notation 5.5. A *weak normal form* of an object A in a category $\mathcal{C}$ is a transitive object in $\mathcal{C}/A$.

A category $\mathcal{C}$ has the *weak normal form property* whenever every object A has a weak normal form. We are now ready to formulate a definition of analytic functors through the existence of weak normal forms for objects in their category of elements.

Notation 5.6. An endofunctor G of Set is an *analytic functor* whenever its category of elements $\mathrm{el}(\mathsf{G})$ has the weak normal form property. Moreover; X is a finite set for every weak normal form $f \in \mathrm{Hom}((X, x), (Y, y))$ in $\mathrm{el}(\mathsf{G})/(Y, y)$.

Example 5.7. The functor mapping X to X^* is analytic, and the weak normal form of a word (X^*, w) is $(\mathrm{letters}(w), w)$ together with the canonical injection from $\mathrm{letters}(w)$ to X. In this specific case, the weak normal forms are in fact initial objects.

Example 5.8. The functor mapping X to $X^{<\alpha}$ is not analytic when $\alpha \geq \omega$, because of the restriction that weak normal forms are defined using finite sets.

Let us now explain how these weak normal forms can be used to define a support associated to the analytic functor, which in turns allows us to build a notion of substructure ordering over initial algebras of analytic functors.

Definition 5.9. *Let* G *be an analytic functor,* (X, x) *be an element in* $\mathrm{el}(\mathsf{G})$ *and* $f \in \mathrm{Hom}((Y, y), (X, x))$ *be a weak normal form in the slice category* $\mathrm{el}(\mathsf{G})/(X, x)$*. We define* $f(Y)$ *as the* support *of* x *in* X*, written* $\mathrm{supp}_X(x)$*.*

Definition 5.10. *Let* G *be an analytic functor and* $(\mu\mathsf{G}, \delta)$ *be an initial algebra of* G*. We say that* $a \in \mu\mathsf{G}$ *is a child of* $b \in \mu\mathsf{G}$ *whenever* $a = b$ *or* $a \in \mathrm{supp}_{\mu\mathsf{G}}(\delta^{-1}(b))$*. The transitive closure of the children relation is called the* substructure ordering *of* $\mu\mathsf{G}$ *and written* $\sqsubseteq$*.*

Example 5.11. The substructure ordering on $\mu\mathsf{G}$ for $\mathsf{G}(X) := \mathbf{1} + \Sigma \times X$ is the suffix ordering of words.

We leverage the notion of substructure ordering to define a suitable topology expander over initial algebras of analytic functors. Note that this ordering appears implicitely in the construction of Hasegawa [17, Definition 2.7].

Definition 5.12. *Let* $\mathsf{G}' \colon \mathsf{Top} \to \mathsf{Top}$ *be a lifting of an analytic functor* G*, and* $(\mu\mathsf{G}, \delta)$ *an initial algebra of* G*. We define* $\mathsf{E}_\Diamond^{\mathsf{G}'}$ *that maps* τ *to the topology generated by* $\uparrow_\sqsubseteq \delta(U)$ *where* $U \in \mathsf{G}'(\mu\mathsf{G}, \tau)$*.*

We say that $\mathsf{lfp}_\tau.\mathsf{E}_\Diamond^{\mathsf{G}'}$ *is the* divisibility topology *over* $\mu\mathsf{G}$*.*

Theorem 5.13. *Let* $\mathsf{G}' \colon \mathsf{Top} \to \mathsf{Top}$ *be a lifting of an analytic functor* G*, and* $(\mu\mathsf{G}, \delta)$ *an initial algebra of* G*. Moreover, we suppose that* G' *preserves inclusions. The map* $\mathsf{E}_\Diamond^{\mathsf{G}'}$ *is a topology expander, hence the divisibility topology is Noetherian.*

As a sanity check, we can apply Theorem 5.13 to the sets of finite words and finite trees, and recover the subword topology and the tree topology that were obtained in an ad-hoc fashion in Section 4. In addition to validating the usefulness of Theorem 5.13, we believe that these are strong indicators that the topologies introduced prior to this work were the right generalisations of Higman's word embedding and Kruskal's tree embedding in a topological setting, and addresses the canonicity issue of the aforementioned topologies.

Lemma 5.14. *The subword topology over* Σ^**, (resp. the tree topology over* $\mathsf{T}(\Sigma)$*) is the divisibility topology associated to the inductive construction of finite words (resp. finite trees).*

5.2 Divisibility Preorders

We are now going to prove that the divisibility topology correctly generalises the corresponding notions on quasi-orderings. In the case of finite words, this translates to the equation $\mathsf{alex}(\leq)^* = \mathsf{alex}(\leq^*)$ [12, Exercise 9.7.30]. We relate the divisibility topology to the divisibility preorder introduced by Hasegawa [17, Definition 2.7].

Theorem 5.15. *Let* G' *the be the lift of an analytic functor respecting Alexandroff topologies, Noetherian spaces, and embeddings. Then, the divisibility topology of* $\mu\mathsf{G}$ *is the Alexandroff topology of the divisibility preorder of* $\mu\mathsf{G}$*, which is a well-quasi-ordering.*

6 Outlook

We have provided a systematic way to place a Noetherian topology over an inductively defined datatype, which is correct with respect to its wqo counterpart whenever it exists. As a byproduct, we obtained a uniform framework that simplifies existing proofs, and serves as an indicator that the pre-existing topologies were the "right generalisations" of their quasi-order counterparts. Let us now briefly highlight some interesting properties of the underlying theory.

Differences with the existing categorical frameworks. The existing categorical frameworks are built around a specific kind of functors [17, 8], while the notion of topology expander only requires talking about one specific set. This allows proving that the ordinal subword topology and the α-branching trees are Noetherian, while these escape both the realm of wqos, and of "well-behaved functors" having finite support functions.

Quasi-analytic functors. In fact, the proof of Theorem 5.13, never relies on the finiteness of the support of an element. This means that the definition of analytic functors can be loosened to allow non finite weak normal forms. We do not know whether this notion of "quasi-analytic functor" already exists in the literature.

Transfinite iterations. As the reader might have noticed, all of the least fixed points considered in this paper are obtained using at most ω steps. This is because the topology expanders that are presented in the paper are all Scott-continuous, i.e., they satisfy the equation $\mathsf{E}(\sup_i \tau_i) = \sup_i \mathsf{E}(\tau_i)$. While Theorem 3.21 does apply to non Scott-continuous topology expanders, we do not know any reasonable example of such expander.

Lack of ordinal invariants. Even though our proof that the ordinal subword topology is Noetherian is shorter than the original one, it actually provides less information. In particular, it does not provide a bound for ordinal rank of the lattice of closed sets (called the *stature* of $\Sigma^{<\alpha}$), whereas a clear bound is provided by the previous approach Goubault-Larrecq et al. [15, Proposition 33]. This limitation already appears in the existing categorical frameworks [17, 8], and we believe that this is inherent to the use of minimal bad sequence arguments.

Acknowledgements. I thank the anonymous reviewers for their helpful suggestions. I thank Jean Goubault-Larrecq and Sylvain Schmitz for their help and support in writing this paper, together with Simon Halfon for his insight on transfinite words.

References

1. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.K.: General decidability theorems for infinite-state systems. Proceedings of LICS'96. pp. 313–321. IEEE (1996). https://doi.org/10.1109/LICS.1996.561359
2. Abdulla, P.A., Jonsson, B.: Verifying networks of timed processes. Proceedings of TACAS'98. Lecture Notes in Computer Science, vol. 1384, pp. 298–312. Springer (1998). https://doi.org/10.1007/BFb0054179
3. Daligault, J., Rao, M., Thomassé, S.: Well-Quasi-Order of Relabel Functions. Order 27(3), 301–315 (2010). https://doi.org/10.1007/s11083-010-9174-0
4. Dershowitz, N., Tzameret, I.: Gap Embedding for Well-Quasi-Orderings. Proceedings of WoLLIC'03. Electronic Notes in Theoretical Computer Science, vol. 84, pp. 80–90. Elsevier (2003). https://doi.org/10.1016/S1571-0661(04)80846-6
5. Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: Ackermannian and Primitive-Recursive Bounds with Dickson's Lemma. Proceedings of LICS'11. pp. 269–278. IEEE (2011). https://doi.org/10.1109/LICS.2011.39
6. Finkel, A., Goubault-Larrecq, J.: Forward analysis for WSTS, part I: completions. Mathematical Structures in Computer Science 30(7), 752–832 (2020). https://doi.org/10.1017/S0960129520000195
7. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science 256(1), 63–92 (2001). https://doi.org/10.1016/S0304-3975(00)00102-X
8. Freund, A.: From Kruskal's Theorem to Friedman's gap condition. Mathematical Structures in Computer Science 30(8), 952–975 (2020). https://doi.org/10.1017/S0960129520000298
9. Gallier, J.H.: Ann. Pure Appl. Logic: Erratum to "What's so special about Kruskal's Theorem and the ordinal γ_0? A survey of some results in proof theory" [53 (1991) 199–260]. Annals of Pure and Applied Logic 89(2), 275 (1997). https://doi.org/10.1016/S0168-0072(97)00043-2
10. Goubault-Larrecq, J.: On Noetherian spaces. Proceedings of LICS'07. pp. 453–462. IEEE (2007). https://doi.org/10.1109/LICS.2007.34
11. Goubault-Larrecq, J.: Noetherian Spaces in Verification. Proceedings of ICALP'10. Lecture Notes in Computer Science, vol. 6199, pp. 2–21. Springer (2010). https://doi.org/10.1007/978-3-642-14162-1_2
12. Goubault-Larrecq, J.: Non-Hausdorff Topology and Domain Theory, New Mathematical Monographs, vol. 22. Cambridge University Press (2013). https://doi.org/10.1017/CBO9781139524438
13. Goubault-Larrecq, J.: Infinitary Noetherian Constructions I. Infinite Words. Colloquium Mathematicum (168), 257–286 (2022). https://doi.org/10.4064/cm8077-4-2021
14. Goubault-Larrecq, J.: Non-Hausdorff Topology and Domain Theory. Electronic supplements to the book – errata. https://projects.lsv.ens-cachan.fr/topology/?page_id=12 (2022)
15. Goubault-Larrecq, J., Halfon, S., Lopez, A.: Infinitary Noetherian Constructions II. Transfinite Words and the Regular Subword Topology (2022), https://doi.org/10.48550/arXiv.2202.05047
16. Goubault-Larrecq, J., Seisenberger, M., Selivanov, V.L., Weiermann, A.: Well Quasi-Orders in Computer Science (Dagstuhl Seminar 16031). Dagstuhl Reports 6(1), 69–98 (2016). https://doi.org/10.4230/DagRep.6.1.69

17. Hasegawa, R.: Two applications of analytic functors. Theoretical Computer Science 272(1), 113–175 (2002). https://doi.org/10.1016/S0304-3975(00)00349-2
18. Higman, G.: Ordering by divisibility in abstract algebras. Proceedings of the London Mathematical Society 3(1), 326–336 (1952). https://doi.org/10.1112/plms/s3-2.1.326
19. Jančar, P.: A note on well quasi-orderings for powersets. Information Processing Letters 72(5), 155–160 (Dec 1999). https://doi.org/10.1016/S0020-0190(99)00149-0
20. Joyal, A.: Foncteurs analytiques et espèces de structures. Combinatoire énumérative. Lecture Notes in Mathematics, vol. 1234, pp. 126–159. Springer (1986). https://doi.org/10.1007/BFb0072514
21. Kříž, I., Thomas, R.: On well-quasi-ordering finite structures with labels. Graphs and Combinatorics 6(1), 41–49 (1990). https://doi.org/10.1007/BF01787479
22. Kruskal, J.B.: The theory of well-quasi-ordering: A frequently discovered concept. Journal of Combinatorial Theory, Series A 13(3), 297–305 (1972). https://doi.org/10.1016/0097-3165(72)90063-5
23. Milner, E.C.: Basic wqo-and bqo-theory. Graphs and order, pp. 487–502. Springer (1985). https://doi.org/10.1007/978-94-009-5315-4_14
24. Nash-Williams, C.St.J.A.: On well-quasi-ordering transfinite sequences. Mathematical Proceedings of the Cambridge Philosophical Society 61(1), 33–39 (1965)
25. Pouzet, M.: Un bel ordre d'abritement et ses rapports avec les bornes d'une multirelation. CR Acad. Sci. Paris Sér. AB 274, A1677–A1680 (1972)
26. Rado, R.: Partial well-ordering of sets of vectors. Mathematika 1(2), 89–95 (1954). https://doi.org/10.1112/S0025579300000565
27. Schmitz, S.: Algorithmic Complexity of Well-Quasi-Orders. Habilitation à diriger des recherches, École normale supérieure Paris-Saclay (2017), https://tel.archives-ouvertes.fr/tel-01663266
28. Schmitz, S., Schnoebelen, P.: Algorithmic Aspects of WQO Theory (2012), https://cel.archives-ouvertes.fr/cel-00727025
29. Segoufin, L., Figueira, D.: Bottom-up automata on data trees and vertical XPath. Logical Methods in Computer Science 13 (2017). https://doi.org/10.23638/LMCS-13(4:5)2017
30. Singh, D., Shuaibu, A.M., Ndayawo: Simplified proof of Kruskal's Tree Theorem. Mathematical Theory and Modeling 3, 93–100 (2013). https://doi.org/10.13140/RG.2.2.12298.39363



An Efficient Cyclic Entailment Procedure in a Fragment of Separation Logic

Quang Loc Le[1](✉) and Xuan-Bach D. Le[2]

[1] Department of Computer Science, University College London, London, UK
loc.le@ucl.ac.uk
[2] School of Computing and Information Systems, University of Melbourne, Melbourne, Australia
bach.le@unimelb.edu.au

Abstract. An efficient entailment proof system is essential to compositional verification using separation logic. Unfortunately, existing decision procedures are either inexpressive or inefficient. For example, Smallfoot is an efficient procedure but only works with hardwired lists and trees. Other procedures that can support general inductive predicates run exponentially in time as their proof search requires back-tracking to deal with a disjunction in the consequent.
This paper presents a decision procedure to derive cyclic entailment proofs for general inductive predicates in polynomial time. Our procedure is efficient and does not require back-tracking; it uses normalisation rules that help avoid the introduction of disjunction in the consequent. Moreover, our decidable fragment is sufficiently expressive: It is based on compositional predicates and can capture a wide range of data structures, including sorted and nested list segments, skip lists with fast-forward pointers, and binary search trees. We implemented the proposal in a prototype tool, called $\texttt{S2S}_{\texttt{Lin}}$, and evaluated it over challenging problems from a recent separation logic competition. The experimental results confirm the efficiency of the proposed system.

Keywords: Cyclic Proofs, Entailment Procedure, Separation Logic.

1 Introduction

Separation logic [20,37] has successfully reasoned about programs manipulating pointer structures. It empowers reusability and scalability through compositional reasoning [6,7]. A compositional verification system relies on bi-abduction technology which is, in turn, based on entailment proof systems. Entailment is defined: Given an antecedent A and a consequent C where A and C are formulas in separation logic, the entailment problem checks whether $A \models C$ is valid. Thus, an efficient decision procedure for entailments is the vital ingredient of an automatic verification system in separation logic.

To enhance the expressiveness of the assertion language, for example, to specify unbounded heaps and interesting pure properties (e.g., sortedness, parent pointers), separation logic is typically combined with user-defined inductive predicates [9,31,35]. In this setting, one key challenge of an entailment procedure is the ability to support induction reasoning over the combination of heaps and data content. The problem of

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 477–497, 2023.
https://doi.org/10.1007/978-3-031-30829-1_23

induction is challenging, especially for an automated inductive theorem prover, where the induction rules are not explicitly stated. Indeed, this problem is undecidable [1].

Developing a sound and complete entailment procedure that could be used for compositional reasoning is not trivial. It is unknown how model-based systems, e.g. [14,15,17,18,22,23], could support compositional reasoning. In contrast, there was evidence that proof-based decision procedures, e.g., Smallfoot [2] and the variant [12], and Cycomp [42], can be extended to solve the bi-abduction problem, which enables compositional reasoning and scalability [7,25]. Smallfoot was the centre of the biabductive procedure deployed in Infer [7], which which greatly impacted academia and industry [13]. Furthermore, Smallfoot is very efficient due to its use of the "exclude-the-middle" rule, which can avoid the proof search over the disjunction in the consequent. However, Smallfoot works for hardwired lists and binary trees only. In contrast, Cycomp, a recent complete entailment procedure, is a cyclic proof system without "exclude-the-middle" and can support general inductive predicates but has double exponential time complexity due to the proof search (and back-tracking) in the consequent.

This paper introduces a cyclic proof system with an "exclude-the-middle"-styled decision procedure for decidable yet expressive inductive predicates. We especially show that our procedure runs in polynomial time when the maximum number of fields of data structures is bounded by a constant. The decidable fragment, `SHLIDe`, contains inductive definitions of compositional predicates and pure properties. These predicates can capture nested list segments, skip lists and trees. The pure properties of small models can model a wide range of common data structures, e.g. a list with fast-forward pointers, sorted nested lists, and binary search trees [22,32]. This fragment is much more expressive than Smallfoot's and is incomparable to Cycomp's [42]: there exist some entailments our system can handle, but Cyccomp could not, and vice versa.

Our procedure is a variant of the cyclic proof system introduced by Brotherston [3,5] and has become one of the leading solutions to induction reasoning in separation logic. Intuitively, a cyclic proof is naturally represented as a tree of statements (entailments in this paper). The leaves are either axioms or nodes linked back to inner nodes; the tree's root is the theorem to be proven, and nodes are connected to one or more children by proof rules. Alternatively, a cyclic proof can be viewed as a tree possibly containing some back-links (a.k.a. cycles, e.g., "C, if B, if C") such that the proof satisfies some global soundness condition. This condition ensures that the proof can be viewed as a proof of *infinite descent*. For instance, for a cyclic entailment proof with inductive definitions, if every cycle contains an unfolding of some inductive predicate, then that predicate is infinitely often reduced into a strictly "smaller" predicate. This infinity is impossible as the semantics of inductive definitions only allows finite steps of unfolding. Hence, that proof path with the cycle can be disregarded.

The proposed system advances Brotherston's system in three ways. First, the proposed proof search algorithm is specialized to `SHLIDe`, which includes "exclude-the-middle" rules and excludes any back-tracking. The existing proof procedures typically search for proof (and back-track) over disjunctive cases generated from unfolding inductive predicates in the RHS of an entailment. To avoid such costly searches, we propose "exclude-the-middle"-styled normalised rules in which the unfolding of inductive predicates in the RHS always produces one disjunct. Therefore, our system is much

more efficient than existing systems. Second, while a standard Brotherston system is incomplete, our proof search is complete in SHLIDe: If it is stuck (i.e., it can not apply any inference rules), then the root entailment is invalid.

Lastly, while the global soundness in [5] must be checked globally and explicitly, every back-link generated in SHLIDe is sound by design. We note that Cycomp, introduced in [42], was the first work to show the completeness of a cyclic proof system. However, in contrast to ours, it did not discuss the global soundness condition, which is the crucial idea attributing to the soundness of cyclic proofs.

Contributions Our primary contributions are summarized as follows.

- We present a novel decision procedure, $\texttt{S2S}_{\texttt{Lin}}$, for the entailment problem in separation logic with inductive definitions of compositional predicates.
- We provide a complexity analysis of the procedure.
- We have implemented the proposal in a prototype tool and tested it with the SL-COMP benchmarks [38,39]. The experimental results show that $\texttt{S2S}_{\texttt{Lin}}$ is effective and efficient compared to state-of-the-art solvers.

Organization The remainder of the paper is organised as follows. Sect. 2 describes the syntax of formulas in fragment SHLIDe. Sect. 3 presents the basics of an "exclude-the-middle" proof system and cyclic proofs. Sect. 4 elaborates on the result, the novel cyclic proof system, including an illustrative example. Sect. 5 discusses soundness and completeness. Sect. 6 presents the implementation and evaluation. Sect. 7 discusses related work. Finally, Sect. 8 concludes the work.

2 Decidable Fragment SHLIDe

Subsection 2.1 presents syntax of separation logic formulae and recursive definitions of linear predicates and local properties. Subsection 2.2 shows semantics.

2.1 Separation Logic Formulas

Concrete heap models assume a fixed finite collection of data structures *Node*, a fixed finite collection of field names *Fields*, a set *Loc* of locations (heap addresses), a set of non-addressable values *Val*, with the requirement that $\mathit{Val} \cap \mathit{Loc} = \emptyset$ (i.e., no pointer arithmetic). null is a special element of *Val*. $\mathbb{Z}$ denotes the set of integers ($\mathbb{Z} \subseteq \mathit{Val}$) and k denotes integer numbers. *Var* an infinite set of variables, $\bar{v}$ a sequence of variables.

Syntax Disjunctive formula Φ, symbolic heaps Δ, spatial formula κ, pure formula π, pointer (dis)equality ϕ, and (in)equality formula α are as follows.

$$\Phi ::= \Delta \mid \Phi \vee \Phi \qquad \Delta ::= \kappa \wedge \pi \mid \exists v.\ \kappa \wedge \pi \qquad \pi ::= \texttt{true} \mid \alpha \mid \neg\pi \mid \pi \wedge \pi$$
$$\kappa ::= \texttt{emp} \mid x \mapsto c(f{:}v, .., f{:}v) \mid \texttt{P}(\bar{v}) \mid \kappa * \kappa \qquad \alpha ::= a{=}a \mid a{\leq}a \qquad a ::= k \mid v$$

where $v \in \mathit{Var}$, $c \in \mathit{Node}$ and $f \in \mathit{Fields}$. Note that we often discard field names f of points-to predicates $x \mapsto c(f{:}v, .., f{:}v)$ and use the short form as $x \mapsto c(\bar{v})$. $v_1 \neq v_2$ is the short form of $\neg(v_1 = v_2)$. E denotes for either a variable or null. $\Delta[E/v]$ denotes the formula obtained from Δ by substituting v by E. *A symbolic heap is referred as a base, denoted as Δ^b, if it does not contain any occurrence of inductive predicates.*

Inductive Definitions We write $\mathcal{P}$ to denote a set of n defined predicates $\mathcal{P}=\{\mathtt{P_1}, ..., \mathtt{P_n}\}$ in our system. Each inductive predicate has following types of parameters: a pair of root and segment defining segment-based linked points-to heaps, reference parameters (e.g., parent pointers, fast-forwarding pointers), transitivity parameters (e.g., singly-linked lists where every heap cell contains the same value a) and pairs of ordering parameters (e.g., trees being binary search trees). An inductive predicate is defined as

$$\begin{array}{l}\mathtt{pred\ P}(r,F,\bar{B},u,sc,tg) \equiv \mathtt{emp}\wedge r{=}F\wedge sc{=}tg\\ \quad\vee\ \exists X_{tl},\bar{Z},sc'.r{\mapsto}c(X_{tl},\bar{p},u,sc')*\kappa'*\mathtt{P}(X_{tl},F,\bar{B},u,sc',tg)\wedge r{\neq}F\wedge sc\diamond sc'\end{array}$$

where r is the root, F the segment, $\bar{B}$ the borders, u the parameter for a transitivity property, sc and tg source and target, respectively, parameters of an order property, $r{\mapsto}c(X_{tl},\bar{p},u,sc')*\kappa'$ the matrix of the heaps, and $\diamond\in\{=,\geq,\leq\}$. (The extension for multiple local properties is straightforward.) Moreover, this definition is constrained by the following three conditions on heap connectivity, establishment, and termination.

Condition C1. In the recursive rule, $\bar{p}=\{\mathtt{null}\}\cup\bar{Z}$. This condition implies that If two variables points to the same heap, their content must be the same. For instance, the following definition of singly-linked lists of even length does not satisfy this condition.

$$\mathtt{pred\ ell}(r,F)\equiv\mathtt{emp}\wedge r{=}F\ \vee\ \exists x_1,X.r{\mapsto}c_1(x_1)*x_1{\mapsto}c_1(X)*\mathtt{ell}(X,F)\wedge r{\neq}F$$

as n_3 and X are not field variables of the node pointed-to by r.

Condition C2. The matrix heap defines nested and connected list segments as:

$$\kappa'{:=}\mathtt{Q}(Z,\bar{U})\mid\kappa'*\kappa'\mid\mathtt{emp}$$

where $Z{\in}\bar{p}$ and $(\bar{U}\setminus\bar{p})\cap Z=\emptyset$. This condition ensures connectivity (i.e. all allocated heaps are connected to the root) and establishment (i.e. every existential quantifier either is allocated or equals to a parameter).

Condition C3. There is no mutual recursion. We define an order $\prec_{\mathcal{P}}$ on inductive predicates as: $\mathtt{P}\prec_{\mathcal{P}}\mathtt{Q}$ if at least one occurrence of predicate $\mathtt{Q}$ appears in the definition of $\mathtt{P}$ and $\mathtt{Q}$ is called a direct sub-term of $\mathtt{P}$. We use $\prec^*_{\mathcal{P}}$ to denote the transitive closure of $\prec_{\mathcal{P}}$.

Several definition examples are shown as follows.

$$\begin{array}{l}\mathtt{pred\ ll}(r,F)\equiv\mathtt{emp}\wedge r{=}F\ \vee\ \exists X_{tl}.r{\mapsto}c_1(X_{tl})*\mathtt{ll}(X_{tl},F)\wedge r{\neq}F\\ \mathtt{pred\ nll}(r,F,B)\equiv\mathtt{emp}\wedge r{=}F\\ \quad\vee\exists X_{tl},Z.r{\mapsto}c_3(X_{tl},Z)*\mathtt{ll}(Z,B)*\mathtt{nll}(X_{tl},F,B)\wedge r{\neq}F\\ \mathtt{pred\ skl1}(r,F)\equiv\mathtt{emp}\wedge r{=}F\ \vee\ \exists X_{tl}.r{\mapsto}c_4(X_{tl},\mathtt{null},\mathtt{null})*\mathtt{skl1}(X_{tl},F)\wedge r{\neq}F\\ \mathtt{pred\ skl2}(r,F)\equiv\mathtt{emp}\wedge r{=}F\\ \quad\vee\exists X_{tl},Z_1.r{\mapsto}c_4(Z_1,X_{tl},\mathtt{null})*\mathtt{skl1}(Z_1,X_{tl})*\mathtt{skl2}(X_{tl},F)\wedge r{\neq}F\\ \mathtt{pred\ skl3}(r,F)\equiv\mathtt{emp}\wedge r{=}F\\ \quad\vee\exists X_{tl},Z_1,Z_2.r{\mapsto}c_4(Z_1,Z_2,X_{tl})*\mathtt{skl1}(Z_1,Z_2)*\mathtt{skl2}(Z_2,X_{tl})*\mathtt{skl3}(X_{tl},F)\wedge r{\neq}F\\ \mathtt{pred\ tree}(r,B)\equiv\mathtt{emp}\wedge r{=}B\\ \quad\vee\ \exists r_l,r_r.r{\mapsto}c_t(r_l,r_r)*\mathtt{tree}(r_l,B)*\mathtt{tree}(r_r,B)\wedge r{\neq}B\end{array}$$

`ll` defines singly-linked lists, `nll` defines lists of acyclic lists, `slk1`, `slk2` and `slk3` define skip-lists. Finally, `tree` defines binary trees. We extend predicate `ll` with transi-

tivity and order parameters to obtain predicate $\texttt{lla}$ and $\texttt{lls}$, respectively, as follows.

$$\texttt{pred lla}(r,F,a) \equiv \texttt{emp}\wedge r{=}F \ \vee\ \exists X_{tl}.r{\mapsto}c_2(X_{tl},a) * \texttt{lla}(X_{tl},F,a)\wedge r{\neq}F$$
$$\texttt{pred lls}(r,F,mi,ma) \equiv \texttt{emp}\wedge r{=}F\wedge ma{=}mi \vee \exists X_{tl},mi_1.r{\mapsto}c_4(X_{tl},mi_1) * \texttt{lls}(X_{tl},F,mi_1,ma)\wedge r{\neq}F \wedge mi{\leq}mi_1$$

Unfolding Given $\texttt{pred}\ \texttt{P}(\bar{t}) \equiv \Phi$ and a formula $\texttt{P}(\bar{v}){*}\Delta$, then unfolding $\texttt{P}(\bar{v})$ means replacing $\texttt{P}(\bar{v})$ by $\Phi[\bar{v}/\bar{t}]$. We annotate a number, called unfolding number, for each occurrence of inductive predicates. Suppose $\exists\bar{w}.r{\mapsto}c(\bar{p}) * \texttt{Q}_1(\bar{v}_1){*}...{*}\texttt{Q}_m(\bar{v}_m) * \texttt{P}(\bar{v}_0)\wedge\pi$ be the recursive rule, then in the unfolded formula, if $\texttt{P}(\bar{v}_0[\bar{v}/\bar{t}])^{k_1}$ and $\texttt{Q}_\texttt{i}(...)^{k_2}$ are direct sub-terms of $\texttt{P}(\bar{v})^k$ like above, then $k_1{=}k{+}1$ and $k_2 = 0$. When it is unambiguous, we discard the annotation of the unfolding number for simplicity.

2.2 Semantics

The program state is interpreted by a pair (s,h) where $s{\in}$*Stacks*, $h{\in}$*Heaps* and stack *Stacks* and heap *Heaps* are defined as:

$$\textit{Heaps} \stackrel{\text{def}}{=} \textit{Loc}{\rightharpoonup_{fin}}(\textit{Node} \rightarrow (\textit{Fields} \rightarrow \textit{Val} \cup \textit{Loc})^m)$$
$$\textit{Stacks} \stackrel{\text{def}}{=} \textit{Var} \rightarrow \textit{Val} \cup \textit{Loc}$$

Note that we assume that every data structure contains at most m fields. Given a formula Φ, its semantics is given by a relation: $s,h \models \Phi$ in which the stack s and the heap h satisfy the constraint Φ. The semantics is shown below

$$\begin{array}{lll}
s,h \models \texttt{emp} & \texttt{iff} & \textit{dom}(h){=}\emptyset \\
s,h \models v{\mapsto}c(f_i : v_i) & \texttt{iff} & \textit{dom}(h){=}\{s(v)\}, h(s(v)){=}g, g(c,f_i){=}s(v_i) \\
s,h \models P(\bar{v}) & \texttt{iff} & (h, s(\bar{v}_1), .., s(\bar{v}_k)) \in [\![P]\!] \\
s,h \models \kappa_1 * \kappa_2 & \texttt{iff} & \exists h_1,h_2\ s.t\ h_1\#h_2, h{=}h_1{\cdot}h_2, , s,h_1 \models \kappa_1 \text{ and } s,h_2 \models \kappa_2 \\
s,h \models \texttt{true} & \texttt{iff} & \text{always} \\
s,h \models \kappa\wedge\pi & \texttt{iff} & s,h \models \kappa \text{ and } s \models \pi \\
s,h \models \exists v.\Delta & \texttt{iff} & \exists\alpha.s[v{\mapsto}\alpha], h \models \Delta \\
s,h \models \Phi_1 \vee \Phi_2 & \texttt{iff} & s,h \models \Phi_1 \text{ or } s,h \models \Phi_2
\end{array}$$

$dom(g)$ is the domain of g, $h_1\#h_2$ denotes disjoint heaps h_1 and h_2 i.e., $\textit{dom}(h_1)\cap \textit{dom}(h_2){=}\emptyset$, and $h_1{\cdot}h_2$ denotes the union of two disjoint heaps. If s is a stack, $v{\in}$*Var*, and $\alpha{\in}$*Val*$\cup$*Loc*, we write $s[v{\mapsto}\alpha] = s$ if $v{\in}\textit{dom}(s)$, otherwise $s[v{\mapsto}\alpha] = s{\cup}\{(v,\alpha)\}$. Semantics of non-heap (pure) formulas is omitted for simplicity. The interpretation of an inductive predicate $\texttt{P}(\bar{t})$ is based on the least fixed point semantics $[\![\texttt{P}]\!]$.

Entailment $\Delta \models \Delta'$ holds iff for all s and h, if $s,h \models \Delta$ then $s,h \models \Delta'$.

3 Entailment Problem & Overview

Throughout this work, we consider the following problem.

PROBLEM:	$\texttt{QF\_ENT}{-}\texttt{SL}_{\texttt{LIN}}$.
INPUT:	$\Delta_a \equiv \kappa_a\wedge\pi_a$ and $\Delta_c \equiv \kappa_c\wedge\pi_c$ where $FV(\Delta_c) \subseteq FV(\Delta_a) \cup \{\texttt{null}\}$.
QUESTION:	Does $\Delta_a \models \Delta_c$ hold?

An entailment, denoted as $\mathtt{e}$, is syntactically formalized as: $\Delta_a \vdash \Delta_c$ where Δ_a and Δ_c are quantifier-free formulas whose syntax are defined in the preceding section.

In Sect. 3.1, we present the basis of an exclude-the-middle proof system and our approach to $\mathtt{QF\_ENT{-}SL_{LIN}}$. In Sect. 3.2, we describe the foundation of cyclic proofs.

3.1 Exclude-the-Middle Proof System

Given a goal $\Delta_a \vdash \Delta_c$, an entailment proof system might derive entailments with a disjunction in the right-hand side (RHS). Such an entailment can be obtained by a proof rule that replaces an inductive predicate by its definition rules. Authors of Smallfoot [2] introduced a normal form and proof rules to prevent such entailments when the predicate are lists or trees. Smallfoot considers the following two scenarios.

- **Case 1** (Exclude-the-middle and Frame): The inductive predicate matches with a points-to predicate in the left-hand side (LHS). For instance, let us consider an entailment which is of the form $\mathtt{e}_1 : x{\mapsto}c(z) * \Delta \vdash \mathtt{ll}(x, y) * \Delta'$, where $\mathtt{ll}$ is singly-linked lists and $\mathtt{ll}(x, y)$ matches with $x{\mapsto}c(z)$ as they have the same root x. A typical proof system might search for proof through two definition rules of predicate $\mathtt{ll}$ (i.e., by unfolding $\mathtt{ll}(x, y)$ into two disjuncts): One includes the base case with $x = y$, and another contains the recursive case with $x \neq y$. Smallfoot prevents such unfolding by excluding the middle in the LHS: It reduces the entailment into two premises: $x{\mapsto}c(z){*}\Delta{\wedge}x = y \vdash \mathtt{ll}(x, y){*}\Delta'$ and $x{\mapsto}c(z){*}\Delta{\wedge}x \neq y \vdash \mathtt{ll}(x, y) * \Delta'$. The first one considers the base case of the list (that is, $\mathtt{ll}(x, x)$) and is equivalent to $x{\mapsto}c(z) * \Delta \wedge x = y \vdash \Delta'$. Furthermore, the second premise checks the inductive case of the list and is equivalent to $\Delta \wedge x \neq y \vdash \mathtt{ll}(x, z) * \Delta'$.
- **Case 2** (Induction proving via hard-wired Lemma). The inductive predicate matches other inductive predicates in the LHS. For example, consider the entailment $\mathtt{e}_2 : \mathtt{ll}(x, z) * \Delta \vdash \mathtt{ll}(x, \mathtt{null}) * \Delta'$. Smallfoot handle $\mathtt{e}_2$ by using a proof rule as the consequence of applying the following hard-wired lemma $\mathtt{ll}(x, z){*}\mathtt{ll}(z, \mathtt{null}) \models \mathtt{ll}(x, \mathtt{null})$ and reduces the entailment to $\Delta \vdash \mathtt{ll}(z, \mathtt{null}) * \Delta'$.

In doing so, Smallfoot does not introduce a disjunction in the RHS. However, as it uses specific lemmas in the induction reasoning, it only works for the hardwired lists.

This paper proposes $\mathtt{S2S_{Lin}}$ as an exclude-the-middle system for user-defined predicates, those in $\mathtt{SHLIDe}$. Instead of using hardwired lemmas, we apply cyclic proofs for induction reasoning. For instance, to discharge the entailment $\mathtt{e}_2$ above, $\mathtt{S2S_{Lin}}$ first unfolds $\mathtt{ll}(x, z)$ in the LHS and obtains two premises:

- $\mathtt{e}_{21} : (\mathtt{emp} \wedge x = z) * \Delta \vdash \mathtt{ll}(x, \mathtt{null}) * \Delta'$; and
- $\mathtt{e}_{22} : (x{\mapsto}c(y) * \mathtt{ll}(y, z) \wedge x \neq z) * \Delta \vdash \mathtt{ll}(x, \mathtt{null}) * \Delta'$

While it reduces $\mathtt{e}_{21}$ to $\Delta[z/x] \vdash \mathtt{ll}(z, \mathtt{null}) * \Delta'[z/x]$, for $\mathtt{e}_{22}$, it further applies the frame rule as in **Case 1** above and obtains $\mathtt{ll}(y, z) * \Delta \wedge x \neq z \vdash \mathtt{ll}(y, \mathtt{null}) * \Delta'$. Then, it makes a backlink between the latter and $\mathtt{e}_2$ and closes this path. Doing so does not introduce disjunctions in the RHS and can handle user-defined predicates.

3.2 Cyclic Proofs

Central to our work is a procedure that constructs a cyclic proof for an entailment. Given an entailment $\Delta \vdash \Delta'$, if our system can derive a cyclic proof, then $\Delta \models \Delta'$. If instead, it is stuck without proof, then $\Delta \models \Delta'$ is not valid.

The procedure includes proof rules, each of which is of the form:

$$\mathtt{PR}_0 \ \frac{\mathtt{e}_1 \quad ... \quad \mathtt{e}_n}{\mathtt{e}} \ \mathtt{cond}$$

where entailment $\mathtt{e}$ (called the conclusion) is reduced to entailments $\mathtt{e}_1$, ..,$\mathtt{e}_n$ (called the premises) through inference rule $\mathtt{PR}_0$ given that the *side condition* $\mathtt{cond}$ holds.

A cyclic proof is a proof tree $\mathcal{T}_i$ which is a tuple $(V, E, \mathcal{C})$ where

- V is a finite set of nodes representing entailments derived during the proof search;
- A directed edge $(\mathtt{e}, \mathtt{PR}, \mathtt{e}') \in E$ (where $\mathtt{e}'$ is a child of $\mathtt{e}$) means that the premise $\mathtt{e}'$ is derived from the conclusion $\mathtt{e}$ via inference rule $\mathtt{PR}$. For instance, suppose that the rule $\mathtt{PR}_0$ above has been applied, then the following n edges are generated: $(\mathtt{e}, \mathtt{PR}_0, \mathtt{e}_1)$, .., $(\mathtt{e}, \mathtt{PR}_0, \mathtt{e}_n)$;
- and $\mathcal{C}$ is a partial relation which captures back-links in the proof tree. If $\mathcal{C}(\mathtt{e}_c \rightarrow \mathtt{e}_b, \sigma)$ holds, then $\mathtt{e}_b$ is linked back to its ancestor $\mathtt{e}_c$ through the substitution σ (where $\mathtt{e}_b$ is referred to as a *bud* and $\mathtt{e}_c$ is referred to as a *companion*). In particular, $\mathtt{e}_c$ is of the form: $\Delta \vdash \Delta'$ and $\mathtt{e}_b$ is of the form: $\Delta_1 \wedge \pi \vdash \Delta'_1$ where $\Delta \equiv \Delta_1 \sigma$ and $\Delta' \equiv \Delta'_1 \sigma$.

A leaf node is marked as closed if it is evaluated as valid (i.e. the node is applied with an axiom), invalid (i.e. no rule can apply), or linked back. Otherwise, it is marked as open. A proof tree is *invalid* if it contains at least one invalid leaf node. It is *pre-proof* if all its leaf nodes are either valid or linked back. Furthermore, a pre-proof is a cyclic proof if a global soundness condition is established in the tree. Intuitively, this condition requires that for every $\mathcal{C}(\mathtt{e}_c \rightarrow \mathtt{e}_b, \sigma)$, there exist inductive predicates $\mathtt{P}(\bar{t_1})$ in $\mathtt{e}_c$ and $\mathtt{Q}(\bar{t_2})$ in $\mathtt{e}_b$ such that $\mathtt{Q}(\bar{t_2})$ is a subterm of $\mathtt{P}(\bar{t_1})$.

Definition 1 (Trace) *Let $\mathcal{T}_i$ be a pre-proof of $\Delta_a \vdash \Delta_c$ and $(\Delta_{a_i} \vdash \Delta_{c_i})_{i \geq 0}$ be a path of $\mathcal{T}_i$. A trace following $(\Delta_{a_i} \vdash \Delta_{c_i})_{i \geq 0}$ is a sequence $(\alpha_i)_{i \geq 0}$ such that each α_i (for all $i \geq 0$) is a subformula of Δ_{a_i} containing predicate $\mathtt{P}(\bar{t})^u$, and either:*

- *α_{i+1} is the subformula occurrence in $\Delta_{a_{i+1}}$ corresponding to α_i in Δ_{a_i}.*
- *or $\Delta_{a_i} \vdash \Delta_{c_i}$ is the conclusion of a left-unfolding rule, $\alpha_i \equiv \mathtt{P}(\bar{t})^u$ is unfolded, and α_{i+1} is a subformula in $\Delta_{a_{i+1}}$ and is the definition rule of $\mathtt{P}(\bar{x})^u[\bar{t}/\bar{x}]$. In this case, i is said to be a* progressing point *of the trace.*

Definition 2 (Cyclic proof) *A pre-proof $\mathcal{T}_i$ of $\Delta_a \vdash \Delta_c$ is a cyclic proof if, for every infinite path $(\Delta_{a_i} \vdash \Delta_{c_i})_{i \geq 0}$ of $\mathcal{T}_i$, there is a tail of the path $p = (\Delta_{a_i} \vdash \Delta_{c_i})_{i \geq n}$ such that there is a trace following p which has infinitely progressing points.*

Suppose that all proof rules are (locally) sound (i.e., if the premises are valid, then the conclusion is valid). The following Theorem shows *global soundness*.

Theorem 1 (Soundness [5]). *If there is a cyclic proof of $\Delta_a \vdash \Delta_c$, then $\Delta_a \models \Delta_c$.*

The proof is by contraction (c.f. [5]). Intuitively, if we can derive a cyclic proof for $\Delta_a \vdash \Delta_c$ and $\Delta_a \not\models \Delta_c$, then the inductive predicates at the progress points are unfolded infinitely often. This infinity contradicts the least semantics of the predicates.

4 Cyclic Entailment Procedure

This section presents our main proposal, the entailment procedure ω-ENT with the proposed inference rules (subsection 4.1), and an illustrative example (subsection 4.2).

4.1 Proof Search

The proof search algorithm ω-ENT is presented in Fig. 1. ω-ENT takes $\mathtt{e}_0$ as input, produces cyclic proofs, and based on that, decides whether the input is `valid` or `invalid`. The idea of ω-ENT is to iteratively reduce $\mathcal{T}_0$ into a sequence of cyclic proof trees $\mathcal{T}_i$, $i \geq 0$. Initially, for every $\mathtt{P}(\bar{v})^k \in \mathtt{e}_0$, k is reset to 0, and $\mathcal{T}_0$ only has $\mathtt{e}_0$ as an open leaf, the root. On line 3, through the procedure $\mathtt{is\_closed}(\mathcal{T}_i)$, ω-ENT chooses an *open* leaf node $\mathtt{e}_i$, and a proof rule PR_i to apply. If $\mathtt{is\_closed}(\mathcal{T}_i)$ returns `valid` (that is, every leaf is applied to an axiom rule or involved in a back-link), ω-ENT returns `valid` on line 4. If it returns `invalid`, then ω-ENT returns `invalid` (one line 5). Otherwise, it tries to link $\mathtt{e}_i$ back to an internal node (on line 6). If this attempt fails, it applies the rule (line 7).

$\omega-$ENT

input: $\mathtt{e}_0$ **output**: `valid` *or* `invalid`

1: $i \leftarrow 0$; $\mathcal{T}_i \leftarrow \mathtt{e}_0$;
2: **while** `true` **do**
3: $(\mathtt{res}, \mathtt{e}_i, PR_i) \leftarrow \mathtt{is\_closed}(\mathcal{T}_i)$;
4: **if** $\mathtt{res}=\mathtt{valid}$ **then return** `valid`;
5: **if** $\mathtt{res}=\mathtt{invalid}$ **then return** `invalid`;
6: **if** $\mathtt{link\_back}_{\mathtt{e}}(\mathcal{T}_i, \mathtt{e}_i) = \mathtt{false}$ **then**
7: $\mathcal{T}_{i+1} \leftarrow \mathtt{apply}(\mathcal{T}_i, \mathtt{e}_i, PR_i)$;
8: $i \leftarrow i+1$;
9: **end**

Fig. 1: Proof tree construction procedure

Note that at each leaf, `is_closed` attempts rules in the following order: normalization rules, axiom rules, and reduction rules. A rule PR_i is chosen if its conclusion can be unified with the leaf through some substitution σ. Then, on line 7, for each premise of PR_i, procedure `apply` creates a new open node and connects the node to $\mathtt{e}_i$ via a new edge. If PR_i is an axiom, procedure `apply` marks $\mathtt{e}_i$ as closed and returns.

Procedure $\mathtt{is\_closed}(\mathcal{T}_i)$ This procedure examines the following three cases.

1. First, if all leaf nodes are marked closed, and none is `invalid`, then `is_closed` returns `valid`.
2. Secondly, `is_closed` returns `invalid` if there exists an open leaf node $\mathtt{e}_i : \ \Delta \vdash \Delta'$ in NF such that one of the four following conditions hold:
   (a) $\mathtt{e}_i$ could not be applied by any inference rule.
   (b) there exists a predicate $op_1(E) \in \Delta$ such that $op_2(E) \notin \Delta'$ and one of the following conditions holds:
   – either $\mathtt{P}(E',E,...)$ or $E' \mapsto c(E,..)$ are on both sides
   – both $\mathtt{P}(E',E,...) \notin \Delta$ and $E' \mapsto c(E,..) \notin \Delta$
   (c) there exists a predicate $op_1(E) \in \Delta'$ such that $G(op_1(E)) \in \Delta$ and $op_2(E) \notin \Delta$.
   (d) there exist $x \mapsto c_1(\bar{v}_1) \in \Delta$, $x \mapsto c_2(\bar{v}_2) \in \Delta'$ such that $c_1 \not\equiv c_2$ or $\bar{v}_1 \not\equiv \bar{v}_2$.
3. Lastly, an open leaf node $\mathtt{e}_i$ could be applied by an inference rule (e.g. PR_i), `is_closed` returns the triple (`unknown`, $\mathtt{e}_i$, PR_i).

In the rest, we discuss the proof rules and the auxiliary procedures in detail.

Normalisation An entailment is in the normal form (NF) if its LHS is in NF. We write $op(E)$ to denote for either $E{\mapsto}c(\bar{v})$ or $\mathtt{P}(E,F,\bar{B},\bar{v})$. Furthermore, the guard $G(op(E))$ is defined by: $G(E{\mapsto}c(\bar{v})) \stackrel{\text{def}}{=} \mathtt{true}$ and $G(\mathtt{P}(E,F,\bar{B},\bar{v})) \stackrel{\text{def}}{=} E{\neq}F$.

Definition 3 (Normal Form) *A formula $\kappa{\wedge}\phi{\wedge}a$ is in normal form if:*

1. $op(E) \in \kappa$ *implies* $G(op(E)) \in \phi$
2. $op(E) \in \kappa$ *implies* $E{\neq}\mathtt{null} \in \phi$
3. $op_1(E_1) * op_2(E_2) \in \kappa$ *implies* $E_1{\neq}E_2 \in \phi$
4. $E_1{=}E_2 \notin \phi$
5. $E{\neq}E \notin \phi$
6. a *is satisfiable*

If Δ is in NF and for any $s, h \models \Delta$, then $dom(h)$ is uniquely defined by s.

The normalisation rules are presented in Fig. 2. Basically, ω-ENT applies these rules to a leaf exhaustively and transforms it into NF before others. Given an inductive predicate $\mathtt{P}(E, F, ...)$, rule ExM excludes the middle by doing case analysis for the predicate between base-case (i.e., $E{=}F$) and recursive-case (i.e., $E{\neq}F$). The normalisation rule $\neq\mathtt{null}$ follows the following facts: $E{\mapsto}c(\_) \Rightarrow E{\neq}\mathtt{null}$ and $\mathtt{P}(E,F,\_){\wedge}E{\neq}F \Rightarrow E{\neq}\mathtt{null}$. Similarly, rule $\neq *$ follows the following facts: $x{\mapsto}\_{*}\mathtt{P}(y,F,\_){\wedge}y{\neq}F \Rightarrow x{\neq}y$, $x{\mapsto}\_{*}y{\mapsto}\_ \Rightarrow x{\neq}y$, and $\mathtt{P_i}(x,F_1,\_){*}\mathtt{P_j}(y,F_2,\_){\wedge}x{\neq}F_1{\wedge}y{\neq}F_2 \Rightarrow x{\neq}y$.

Axiom and Reduction Axiom rules include Emp, Inconsistency and Id, presented in Fig. 3. If each of these rules is applied to a leaf node, the node is evaluated as valid and marked as closed. The remaining ones in Fig. 3 are reduction rules.

For simplicity, the unfoldings in rules Frame, RInd, and LInd are applied with the following definition of inductive predicates:

$$\begin{array}{l}\mathtt{P}(x,F,\bar{B},u,sc,tg) \equiv \mathtt{emp}{\wedge}x{=}F{\wedge}sc{=}tg\\ \quad \vee\, \exists X,sc',d_1,d_2.x{\mapsto}c(X,d_1,d_2,u,sc){*}\mathtt{Q_1}(d_1,B){*}\mathtt{Q_2}(d_2,X){*}\mathtt{P}(X,F,\bar{B},u,sc',tg){\wedge}\pi_0\end{array}$$

where $B{\in}\bar{B}$, the matrix κ' contains two nested predicates Q_1 and Q_2, and the heap cell $c \in$ *Node* is defined as $\mathtt{data}\ c\{c\ next; c_1\ down_1; c_2\ down_2; \tau_s\ scdata; \tau_u\ udata\}$ where $c_1, c_2{\in}$*Node*, $down_1$ and $down_2$ fields are for the nested predicates in the matrix

$$\mathtt{Subst}\ \frac{\Delta[E/x] \vdash \Delta'[E/x]}{\Delta{\wedge}x{=}E \vdash \Delta'} \qquad \mathrm{ExM}\ \frac{\begin{array}{c}\Delta{\wedge}E_1{=}E_2 \vdash \Delta'\\ \Delta{\wedge}E_1{\neq}E_2 \vdash \Delta'\end{array}}{\Delta \vdash \Delta'}\ {\scriptstyle E_1{=}E_2,\, E_1{\neq}E_2{\notin}\pi\ \&\ FV(E_1,E_2) \subseteq (FV(\Delta){\cup}FV(\Delta'))^S}$$

$$=\mathrm{L}\ \frac{\Delta \vdash \Delta'}{\Delta{\wedge}E{=}E \vdash \Delta'} \qquad \mathrm{LBase}\ \frac{(\kappa{\wedge}\pi)[tg/sc] \vdash \Delta'[tg/sc]}{\mathtt{P}(E,E,\bar{B},u,sc,tg){*}\kappa{\wedge}\pi \vdash \Delta'}$$

$$\neq\mathtt{null}\ \frac{op(E){*}\kappa{\wedge}\pi{\wedge}G(op(E)){\wedge}E{\neq}\mathtt{null} \vdash \Delta'}{op(E){*}\kappa{\wedge}\pi{\wedge}G(op(E)) \vdash \Delta'}\ E{\neq}\mathtt{null}{\notin}\pi$$

$$\neq *\ \frac{op_1(E_1){*}op_2(E_2){*}\kappa{\wedge}\pi{\wedge}E_1{\neq}E_2 \vdash \Delta'}{op_1(E_1){*}op_2(E_2){*}\kappa{\wedge}\pi \vdash \Delta'}\ E_1{\neq}E_2{\notin}\pi \text{ and } G(op_1(E_1)), G(op_2(E_2)) \in \pi$$

Fig. 2: Normalization rules

$$\text{Id}\ \frac{}{\Delta \wedge \pi \vdash \Delta} \qquad \text{Emp}\ \frac{}{\texttt{emp}\wedge\pi \vdash \texttt{emp}\wedge\texttt{true}} \qquad \text{Inconsistency}\ \frac{}{\kappa\wedge\pi \vdash \Delta}\ \pi \models \texttt{false}$$

$$=\text{R}\ \frac{\Delta \vdash \Delta'}{\Delta \vdash \Delta'\wedge E{=}E} \qquad \text{Hypothesis}\ \frac{\Delta\wedge\pi \vdash \Delta'}{\Delta\wedge\pi \vdash \Delta'\wedge\pi'}\ \pi \models \pi' \qquad \text{RBase}\ \frac{\Delta \vdash \Delta' \wedge tg{=}sc}{\Delta \vdash \texttt{P}(E,E,\bar{B},u,sc,tg)*\Delta'}$$

$$*\ \frac{\kappa_1\wedge\pi \vdash \kappa_2 \quad \kappa\wedge\pi \vdash \kappa'\wedge\pi'}{\kappa_1*\kappa\wedge\pi \vdash \kappa_2*\kappa' \wedge \pi'}\ \begin{array}{l}\texttt{roots}(\kappa_1)\cap\texttt{roots}(\kappa)=\emptyset\ \&\ FV(\kappa_2){\subseteq}FV(\kappa_1\wedge\pi){\cup}\{\texttt{null}\}\\ \qquad\qquad \&\ FV(\kappa'){\subseteq}FV(\kappa\wedge\pi){\cup}\{\texttt{null}\}\end{array}$$

$$\text{Frame}\ \frac{\begin{array}{c}\texttt{Q}_1(E_1,B)^0{*}\texttt{Q}_2(E_2,X)^0{*}\texttt{P}(X,F,\bar{B},u,sc',tg)^k{*}\Delta_1\wedge x{\neq}F_3\wedge\pi_0\\ \vdash \texttt{Q}(x,F_3,\bar{B},u,sc,tg_2){*}\kappa_2\wedge\pi_2\end{array}}{\texttt{P}(x,F,\bar{B},u,sc,tg)^k{*}\Delta_1\wedge x{\neq}F_3 \vdash x{\mapsto}c(X,E_1,E_2,u,sc'){*}\kappa_2\wedge\pi_2}\ x{\mapsto}c(\_){\notin}\kappa_2$$

$$\text{RInd}\ \frac{\begin{array}{c}x{\mapsto}c(X,E_1,E_2,u,sc'){*}\kappa_1\wedge\pi_1\wedge x{\neq}F\\ \vdash x{\mapsto}c(X,E_1,E_2,u,sc'){*}\texttt{Q}_1(E_1,B){*}\texttt{Q}_2(E_2,X){*}\texttt{P}(X,F,\bar{B},u,sc',tg){*}\kappa_2\wedge\pi_2\wedge\pi_0\end{array}}{x{\mapsto}c(X,E_1,E_2,u,sc'){*}\kappa_1\wedge\pi_1\wedge x{\neq}F \vdash \texttt{P}(x,F,\bar{B},u,sc,tg){*}\kappa_2\wedge\pi_2}\ \dagger$$

$$\text{LInd}\ \frac{\begin{array}{c}x{\mapsto}c(X,E_1,E_2,u,sc'){*}\texttt{Q}_1(E_1,B)^0{*}\texttt{Q}_2(E_2,X)^0{*}\texttt{P}(X,F,\bar{B},u,sc',tg)^{k+1}{*}\Delta_1\wedge x{\neq}F_3\wedge\pi_0\\ \vdash \texttt{Q}(x,F_3,\bar{B},u,sc,tg_2){*}\kappa_2\wedge\pi_2\end{array}}{\texttt{P}(x,F,\bar{B},u,sc,tg)^k{*}\Delta_1\wedge x{\neq}F_3 \vdash \texttt{Q}(x,F_3,\bar{B},u,sc,tg_2){*}\kappa_2\wedge\pi_2}\ \sharp$$

Fig. 3: Reduction rules (where $\sharp$: $\texttt{P}(x,F,\bar{B},u,sc,tg){\notin}\kappa_2$, $\dagger$: $x{\mapsto}c(X,E_1,E_2,u,sc'){\notin}\kappa_2$)

heaps, the *udata* field is for the transitivity data, and the *scdata* field is for ordering data. The rules for the general form of the matrix heaps κ' are presented in [28].

$=$R and Hypothesis eliminate pure constraints in the RHS. In rule $*$, $\texttt{roots}(\kappa)$ is defined inductively as: $\texttt{roots}(\texttt{emp}){\equiv}\{\}$, $\texttt{roots}(r{\mapsto}\_){\equiv}\{r\}$, $\texttt{roots}(P(r,F,..)){\equiv}\{r\}$ and $\texttt{roots}(\kappa_1{*}\kappa_2) \equiv \texttt{roots}(\kappa_1){\cup}\texttt{roots}(\kappa_2)$. This rule is applied in three ways. First, it is applied into an entailment which is of the form $\kappa\wedge\pi \vdash \kappa\wedge\pi'$. It matches and discards the identified heap predicates between the two sides to generate a premise with empty heaps. As a result, this premise may be applied with the axiom rule EMP. Secondly, it is applied to an entailment of the form $x_i{\mapsto}c_i(\bar{v}_i){*}...{*}x_n{\mapsto}c_n(\bar{v}_n)\wedge\pi \vdash \kappa'\wedge\pi'$. For each points-to predicate $x_i{\mapsto}c_i(\bar{v}_i){\in}\kappa'$, ω-ENT searches for one points-to predicate $x_j{\mapsto}c_j(\bar{v}_j)$ in the LHS such that $x_j{\mapsto}c_j(\bar{v}_j) \equiv x_i{\mapsto}c_i(\bar{v}_i)$. Lastly, it is applied into an entailment that is of the form $\Delta_1 * \Delta \vdash \Delta_2 * \Delta'$ where either $\Delta_1 \vdash \Delta_2$ or $\Delta \vdash \Delta'$ could be linked back into an internal node.

In RInd, for each occurrence of inductive predicates $\texttt{P}(r,F,\bar{B},u,sc,tg)$ in κ', ω-ENT searches for a points-to predicate $r{\mapsto}\_$. If any of these searches fail, ω-ENT decides the conclusion as invalid. Rule LInd unfolds the inductive predicates in the LHS. Every LHS of entailments in this rule also captures the unfolding numbers for the subterm relationship and generates the progressing point in the cyclic proofs afterwards. These numbers are essential for our system to construct cyclic proofs. This rule is applied in a *depth-first* manner, i.e., if there are more than one occurrences of inductive predicates in the LHS that could be applied by this rule, the one with the greatest unfolding number is chosen. We emphasise that the last five rules still work well when the predicate in the RHS contains only a subset of the local properties wrt. the predicate in the LHS.

Back-Link Generation Procedure $\texttt{link\_back}_{\texttt{e}}$ generates a back-link as follows. In a pre-proof, given a path containing a back-link, say $\texttt{e}_1, \texttt{e}_2, .., \texttt{e}_m$ where $\texttt{e}_1$ is a companion and $\texttt{e}_m$ a bud, then $\texttt{e}_1$ is in NF and of the following form:

- $\texttt{e}_1 \equiv \texttt{P}(x,F,\bar{B},u,sc,tg)^k * \kappa \wedge \pi \wedge x \neq F \wedge x \neq \texttt{null} \vdash \texttt{Q}(x,F_2,\bar{B},u,sc,tg_2) * \kappa' \wedge \pi'$.
- $\texttt{e}_2$ is obtained from applying $\texttt{LInd}$ into $\texttt{e}_1$. $\texttt{e}_2$ is of the form:

$$x \mapsto c(X,\bar{p},,u,sc) * \kappa' * \texttt{P}(X,F,\bar{B},u,sc',tg)^{k+1} * \kappa \wedge \pi \wedge x \neq F \wedge x \neq \texttt{null} \wedge \pi_1 \vdash \texttt{Q}(x,F_2,\bar{B},u,sc,tg_2) * \kappa' \wedge \pi'$$

  We remark that $sc \diamond sc' \in \pi_1$, and if $k \geq 1$, then $sc_i \diamond sc \in \pi$
- $\texttt{e}_3$, .., $\texttt{e}_{m-4}$ are obtained from applications of normalisation rules to normalise the LHS of $\texttt{e}_2$ due to the presence of κ'. As the roots of inductive predicates in κ' are fresh variables, the applications of the normalization rules above do not affect the RHS of $\texttt{e}_2$. That means the RHS of $\texttt{e}_3$, .., and $\texttt{e}_{m-4}$ are the same as that of $\texttt{e}_2$. As a result, $\texttt{e}_{m-4}$ is of the form:

$$x \mapsto c(X,\bar{p},,u,sc) * \kappa_1'' * \texttt{P}(X,F,\bar{B},u,sc',tg)^{k+1} * \kappa \wedge \pi \wedge x \neq F \wedge x \neq \texttt{null} \wedge \pi_1 \wedge \pi_2 \vdash \texttt{Q}(x,F_2,\bar{B},u,sc,tg_2) * \kappa' \wedge \pi'$$

  where κ_1'' may be $\texttt{emp}$ and π_2 is a conjunction of disequalities coming from $\texttt{ExM}$.
- $\texttt{e}_{m-3}$ is obtained from the application of $\texttt{ExM}$ over x and F_2 and of the form:

$$x \mapsto c(X,\bar{p},,u,sc) * \kappa_1'' * \texttt{P}(X,F,\bar{B},u,sc',tg)^{k+1} * \kappa \wedge \pi \wedge x \neq F \wedge x \neq \texttt{null} \wedge \pi_1 \wedge \pi_2 \wedge x \neq F_2 \vdash \texttt{Q}(x,F_2,\bar{B},u,sc,tg_2) * \kappa' \wedge \pi'$$

  (For the case $x{=}F_2$, the rule $\texttt{ExM}$ is kept applying until either $F \equiv F_2$, that is, two sides are reaching the end of the same heap segment, or it is stuck.)
- $\texttt{e}_{m-2}$ is obtained from the application of $\texttt{RInd}$ and is of the form:

$$x \mapsto c(X,\bar{p},,u,sc) * \kappa_1'' * \texttt{P}(X,F,\bar{B},u,sc',tg)^{k+1} * \kappa \wedge \pi \wedge x \neq F \wedge x \neq \texttt{null} \wedge \pi_1 \wedge \pi_2 \wedge x \neq F_2 \vdash x \mapsto c(X,\bar{p},u,sc) * \kappa_2'' * \texttt{Q}(X,F_2,\bar{B},u,sc',tg_2) * \kappa' \wedge \pi' \wedge \pi_2'$$

- $\texttt{e}_{m-1}$ is obtained from the application of the $\texttt{Hypothesis}$ to eliminate π_2' (otherwise, it is stuck) and is of the form:

$$x \mapsto c(X,\bar{p},,u,sc) * \kappa_1'' * \texttt{P}(X,F,\bar{B},u,sc',tg)^{k+1} * \kappa \wedge \pi \wedge x \neq F \wedge x \neq \texttt{null} \wedge \pi_1 \wedge \pi_2 \wedge x \neq F_2 \vdash x \mapsto c(X,\bar{p},u,sc) * \kappa_2'' * \texttt{Q}(X,F_2,\bar{B},u,sc',tg_2) * \kappa' \wedge \pi'$$

- $\texttt{e}_m$ is obtained from the application of $*$ and is of the form:

$$\texttt{P}(X,F,\bar{B},u,sc',tg)^{k+1} * \kappa \wedge \pi \wedge x \neq F \wedge x \neq \texttt{null} \wedge \pi_1 \wedge \pi_2 \wedge x \neq F_2 \vdash \texttt{Q}(X,F_2,\bar{B},u,sc',tg_2) * \kappa' \wedge \pi'$$

When $k \geq 1$, it is always possible to link $\texttt{e}_m$ back to $\texttt{e}_1$ through the substitution is $\sigma \equiv [x/X, sc/sc']$ after weakening some pure constraints in its LHS.

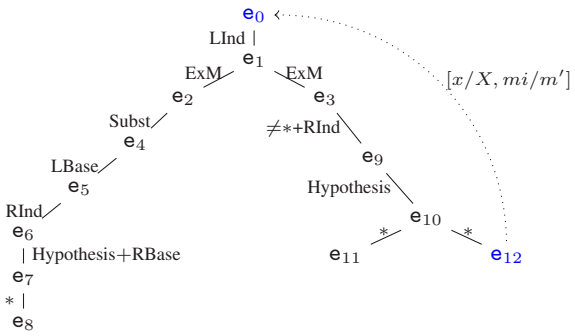


Fig. 4: Cyclic Proof of $\texttt{lls}(x,\texttt{null},mi,ma)^0 \wedge x \neq \texttt{null} \vdash \texttt{llb}(x,\texttt{null},mi)$.

4.2 Illustrative Example

We illustrate our system through the following example:

$$\texttt{e}_0\colon \texttt{lls}(x,\texttt{null},mi,ma)^0 \wedge x \neq \texttt{null} \vdash \texttt{llb}(x,\texttt{null},mi)$$

where the sorted linked-list `lls` (mi is the minimum value and ma is the maximum value) is defined in Sect. 2.1 and `llb` define singly-linked lists whose values are greater than or equal to a constant number. Particularly, predicate `llb` is defined as follows.

$$\begin{aligned}&\texttt{pred llb}(r,F,b) \equiv \texttt{emp} \wedge r = F\\ &\quad \vee \exists X_{tl}, d.r \mapsto c_4(X_{tl},d) * \texttt{llb}(X_{tl},F,b) \wedge r \neq F \wedge b \leq d\end{aligned}$$

Since the LHS is stronger than the RHS, this entailment is valid. Our system could generate the cyclic proof (shown in Fig. 4) to prove the validity of $\texttt{e}_0$. In the following, we present step-by-step to show how the proof was created. Firstly, $\texttt{e}_0$, which is in NF, is applied with rule `LInd` to unfold predicate $\texttt{lls}(x,\texttt{null},mi,ma)^0$ and obtain $\texttt{e}_1$ as:

$$\texttt{e}_1\colon x \mapsto c_4(X, m') * \texttt{lls}(X,\texttt{null},m',ma)^1 \wedge x \neq \texttt{null} \wedge mi \leq m' \vdash \texttt{llb}(x,\texttt{null},mi)$$

We remark that the unfolding number of the recursive predicate `lls` in the LHS is increased by 1. Next, our system normalizes $\texttt{e}_1$ by applying rule `ExM` into X and `null` to generate two children, $\texttt{e}_2$ and $\texttt{e}_3$, as follows.

$$\begin{aligned}&\texttt{e}_2\colon x \mapsto c_4(X, m') * \texttt{lls}(X,\texttt{null},m',ma)^1 \wedge x \neq \texttt{null} \wedge mi \leq m' \wedge X = \texttt{null}\\ &\qquad \vdash \texttt{llb}(x,\texttt{null},mi)\\ &\texttt{e}_3\colon x \mapsto c_4(X, m') * \texttt{lla}(X,\texttt{null},m',ma)^1 \wedge x \neq \texttt{null} \wedge mi \leq m' \wedge X \neq \texttt{null}\\ &\qquad \vdash \texttt{llb}(x,\texttt{null},mi)\end{aligned}$$

For the left child, it applies normalization rules to obtain $\texttt{e}_4$ (substitute X by `null`) and then $\texttt{e}_5$, by `LBase` to unfold $\texttt{lls}(\texttt{null},\texttt{null},m',ma)^1$ to the base case, as:

$$\begin{aligned}&\texttt{e}_4\colon x \mapsto c_4(\texttt{null}, m') * \texttt{lls}(\texttt{null},\texttt{null},m',ma)^1 \wedge x \neq \texttt{null} \wedge mi \leq m' \vdash \texttt{llb}(x,\texttt{null},mi)\\ &\texttt{e}_5\colon x \mapsto c_4(\texttt{null}, ma) \wedge x \neq \texttt{null} \wedge mi \leq ma \vdash \texttt{llb}(x,\texttt{null},mi)\end{aligned}$$

Now, $\mathtt{e}_5$ is in NF. $\mathtt{S2S}_{\mathtt{Lin}}$ applies `RInd` and then `RBase` to `llb` in the RHS as:

$$\begin{aligned}\mathtt{e}_6\colon\ & x{\mapsto}c_4(\mathtt{null}, ma) \wedge x{\neq}\mathtt{null} \wedge mi{\leq}ma\\ & \vdash x{\mapsto}c_4(\mathtt{null}, ma) * \mathtt{llb}(\mathtt{null},\mathtt{null},mi) \wedge mi{\leq}ma\\ \mathtt{e}_{6'}\colon\ & x{\mapsto}c_4(\mathtt{null}, ma) \wedge x{\neq}\mathtt{null} \wedge mi{\leq}ma \vdash x{\mapsto}c_4(\mathtt{null}, ma){\wedge}mi{\leq}ma\end{aligned}$$

After that, as $mi{\leq}ma \Rightarrow mi{\leq}ma$, $\mathtt{e}_{6'}$ is applied with `Hypothesis` to obtain $\mathtt{e}_7$.

$$\mathtt{e}_7\colon x{\mapsto}c_4(\mathtt{null}, ma) \wedge x{\neq}\mathtt{null} \wedge mi{\leq}ma \vdash x{\mapsto}c_4(\mathtt{null}, ma)$$

As the LHS of $\mathtt{e}_7$ is in NF and a base formula, it is sound and complete to apply rule $*$ to have $\mathtt{e}_8$ as $\mathtt{emp} \wedge x{\neq}\mathtt{null} \wedge mi{\leq}ma \vdash \mathtt{emp}$. By `Emp`, $\mathtt{e}_8$ is decided as `valid`. For the right branch of the proof, $\mathtt{e}_3$ is applied with rule $\neq *$ and then `RInd` to obtain $\mathtt{e}_9$:

$$\begin{aligned}\mathtt{e}_9\colon\ & x{\mapsto}c_4(X, m'){*}\mathtt{lls}(X,\mathtt{null},m',ma)^1 \wedge x{\neq}\mathtt{null} \wedge mi{\leq}m' \wedge X{\neq}\mathtt{null} \wedge x{\neq}X\\ & \vdash x{\mapsto}c_4(X, m'){*}\mathtt{llb}(X,\mathtt{null},mi){\wedge}mi{\leq}m'\end{aligned}$$

Then, $\mathtt{e}_9$ is applied with `Hypothesis` to eliminate the pure constraint in the RHS:

$$\begin{aligned}\mathtt{e}_{10}\colon\ & x{\mapsto}c_4(X, m'){*}\mathtt{lls}(X,\mathtt{null},m',ma)^1 \wedge x{\neq}\mathtt{null} \wedge mi{\leq}m' \wedge X{\neq}\mathtt{null} \wedge x{\neq}X\\ & \vdash x{\mapsto}c_4(X, m'){*}\mathtt{llb}(X,\mathtt{null},mi)\end{aligned}$$

$\mathtt{e}_{10}$ is then applied the rule $*$ to obtain $\mathtt{e}_{11}$ and $\mathtt{e}_{12}$ as follows.

$$\begin{aligned}&\mathtt{e}_{11}\colon x{\mapsto}c_4(X, m') \vdash x{\mapsto}c_4(X, m')\\ &\mathtt{e}_{12}\colon \mathtt{lls}(X,\mathtt{null},m',ma)^1 \wedge x{\neq}\mathtt{null} \wedge mi{\leq}m' \wedge X{\neq}\mathtt{null} \wedge x{\neq}X \vdash \mathtt{llb}(X,\mathtt{null},mi)\end{aligned}$$

$\mathtt{e}_{11}$ is valid by `Id`. $\mathtt{e}_{12}$ is successfully linked back to $\mathtt{e}_0$ to form a pre-proof as

$$(\mathtt{lls}(X,\mathtt{null},m',ma)^1{\wedge}X{\neq}\mathtt{null})[x/X, mi/m'] \vdash \mathtt{llb}(X,\mathtt{null},mi)[x/X, mi/m']$$

is identical to $\mathtt{e}_0$. Since $\mathtt{lls}(X,\mathtt{null},m',ma)^1$ in $\mathtt{e}_{12}$ is the subterm of $\mathtt{lls}(x,\mathtt{null},mi,ma)^0$ in $\mathtt{e}_0$, our system decided that $\mathtt{e}_0$ is valid with the cyclic proof presented in Fig. 4.

5 Soundness, Completeness, and Complexity

We describe the soundness, termination, and completeness of ω-ENT. First, we need to show the invariant about the quantifier-free entailments of our system.

Corollary 1. *Every entailment derived from ω-ENT is quantifier-free.*

The following lemma shows the soundness of the proof rules.

Lemma 1 (Soundness). *For each proof rule, the conclusion is valid if all premises are valid.*

As every backlink generated contains at least one pair of inductive predicate occurrences in a subterm relationship, the global soundness condition holds in our system.

Lemma 2 (Global Soundness). *A pre-proof derived is indeed a cyclic proof.*

The termination relies on the number of premises/entailments generated by $*$. As the number of inductive symbols and their arities are finite, there is a finite number of equivalence classes of these entailments in which any two entailments in the same class are equivalent under some substitution and linked back together. Therefore, the number of premises generated by the rule $*$ is finite, considering the back-links generation.

Lemma 3. *ω-ENT terminates.*

In the following, we show the complexity analysis. First, we show that every occurrence of inductive predicates in the LHS is unfolded at most two times.

Lemma 4. *Given any entailment $\mathtt{P}(\bar{v})^k * \Delta_a \vdash \Delta_c$, $0 \leq k \leq 2$.*

Let n be the maximum number of predicates (both inductive predicates and points-to predicates) among the LHS of the input and the definitions in $\mathcal{P}$, and m be the maximum number of fields of data structures. Then, the complexity is defined as follows.

Proposition 1 (Complexity). $\mathtt{QF\_ENT{-}SL_{LIN}}$ *is* $\mathcal{O}(n \times 2^m + n^3)$.

If m is bounded by a constant, the complexity becomes polynomial in time.

Our completeness proofs are shown in two steps. First, we show the proofs for an entailment whose LHS is a base formula. Second, we show the correctness when the LHS contains inductive predicates. In the following, we first define the base formulas of the LHS derived by ω-ENT from occurrences of inductive predicates. Based on that, we define bad models to capture counter-models of invalid entailments.

Definition 4 (SHLIDe Base) *Given κ, define $\overline{\kappa}$ as follows.*

$$\overline{\mathtt{P}(E,F,\bar{B},u,sc,tg)} \stackrel{def}{=} E{\mapsto}c(F,E_1,E_2,u,tg) * \overline{\mathtt{Q}_1(E_1,B)} * \overline{\mathtt{Q}_2(E_2,F)} \wedge \pi_0$$

$$\overline{E{\mapsto}c(\bar{v})} \stackrel{def}{=} E{\mapsto}c(\bar{v}) \qquad \overline{\mathtt{emp}} \stackrel{def}{=} \mathtt{emp} \qquad \overline{\kappa_1 * \kappa_2} \stackrel{def}{=} \overline{\kappa_1} * \overline{\kappa_2}$$

The definition for general predicates with arbitrary matrix heaps is presented in [28]. As $\mathcal{P}$ does not include mutual recursion (Condition **C3**), the definition above terminates in a finite number of steps. In a pre-proof, these SHLIDe base formulas of the LHS are obtained once every inductive predicate has been unfolded.

Lemma 5. *If $\kappa \wedge \pi$ is in NF, then $\overline{\kappa} \wedge \pi$ is in NF, and $\overline{\kappa} \wedge \pi \vdash \kappa$ is valid.*

In other words, $\overline{\kappa} \wedge \pi$ is an under-approximation of $\kappa \wedge \pi$; invalidity of $\overline{\kappa} \wedge \pi \vdash \Delta'$ implies invalidity of $\kappa \wedge \pi \vdash \Delta'$.

Definition 5 (Bad Model) *The bad model for $\overline{\kappa} \wedge \phi \wedge a$ in NF is obtained by assigning*

- *a distinct non-*`null` *value to each variable in FV$(\overline{\kappa} \wedge \phi)$; and*
- *a value to each variable in FV(a) such that a is satisfiable.*

Lemma 6. *1. For every proof rule except the rule $*$, all premises are valid only if the conclusion is valid.*

2. For the rule $*$, where the conclusion is of the form $\Delta^b \vdash \kappa'$, all premises are valid only if the conclusion is valid and Δ^b is in NF.

The following lemma states that the correctness of the procedure `is_closed` for cases 2(b-d).

Lemma 7 (Stuck Invalidity). *Given* $\kappa \wedge \pi \vdash \Delta'$ *in NF, it is* `invalid` *if the procedure* `is_closed` *returns* `invalid` *for cases 2(b-d).*

A bad model of the $\overline{\kappa} \wedge \pi$ is a counter-model. Cases 2b) and 2c) show that the heaps of bad models are not connected, and thus accordingly to conditions **C1** and **C2**, any model of the LHS could not be a model of the RHS. Case 2d) shows that heaps of the two sides could not be matched. We next show the correctness of Case 2(a) of the procedure `is_closed`, and invalidity is preserved during the proof search in ω-`ENT`.

Proposition 2 (Invalidity Preservation). *If* ω-`ENT` *is stuck, the input is invalid.*

In other words, if ω-`ENT` returns invalid, we can construct a bad model.

Theorem 2. `QF_ENT`$-\mathtt{SL}_{\mathtt{LIN}}$ *is decidable.*

6 Implementation and Evaluation

We implement $\mathtt{S2S}_{\mathtt{Lin}}$ using OCaml. This implementation is an instantiation of a general framework for cyclic proofs. We utilize the cyclic proof systems to derive bases for inductive predicates shown in [24] to discharge satisfiability of separation logic formulas. We use the solver presented in [29,31] for those formulas beyond this fragment. We also develop a built-in solver for discharging equalities.

We evaluated $\mathtt{S2S}_{\mathtt{Lin}}$ to show that i) it can discharge problems in `SHLIDe` effectively; and ii) its performance is compatible with state-of-the-art solvers. The evaluation of $\mathtt{S2S}_{\mathtt{Lin}}$ is provided as a companion artifact [27].

Experiment settings We have evaluated $\mathtt{S2S}_{\mathtt{Lin}}$ on entailment problems taken from SL-COMP benchmarks [38], a competition of separation logic solvers. We take 356 problems (out of 983) in two divisions of the competition, *qf_shls_entl* and *qf_shlid_entl*, and one new division, *qf_shlid2_entl*. All these problems semantically belong to our decidable fragment, and their syntax is written in SMT 2.6 format [39].

- Division *qf_shls_entl* includes 296 entailment problems, 122 `invalid` problems and 174 `valid` problems, with only singly-linked lists. The authors in [33] randomly generated them
- Division *qf_shlid_entl* contains 60 entailment problems which the authors in [15] handcrafted. They include singly-linked lists, doubly-linked lists, lists of singly-linked lists, or skip lists. Furthermore, the system of inductive predicates must satisfy the following condition: For two different predicates `P`, `Q` in the system of definitions, either $\mathtt{P} \prec^*_{\mathcal{P}} \mathtt{Q}$ or $\mathtt{Q} \prec^*_{\mathcal{P}} \mathtt{P}$.
- In the third division, we introduce new benchmarks, with 27 problems, beyond the above two divisions. In particular, every system of predicate definitions includes two predicates, `P` and `Q`, that are semantically equivalent. We have submitted this division to the Github repository of SL-COMP.

Table 1: Experimental results

Tool	*qf_shls_entl*			*qf_shlid_entl*			*qf_shlid2_entl*		
	`invalid` (122)	`valid` (174)	Time (296)	`invalid` (24)	`valid` (36)	Time (60)	`invalid` (14)	`valid` (13)	Time (27)
SLS	12	174	507m42s	2	35	133m28s	0	11	97m54s
`Spen`	122	174	10.78s	14	13	3.44s	8	2	1.69s
$\texttt{Cyclist}_{\texttt{SL}}$	0	58	1520m5s	0	24	360m38s	0	3	240m3s
Harrsh	39	116	425m19s	18	27	53m56s	8	7	156m45s
`Songbird`	12	174	237m25s	2	35	40m38s	0	12	47m11s
$\texttt{S2S}_{\texttt{Lin}}$	122	174	6.22s	24	36	0.96s	14	13	1.20s

To evaluate $\texttt{S2S}_{\texttt{Lin}}$'s performance, we compared it with the state-of-the-art tools such as $\texttt{Cyclist}_{\texttt{SL}}$ [5], `Spen` [15], `Songbird` [40], SLS [41] and Harrsh [23]. We omitted Cyclist-comp [42], as these benchmarks are beyond its decidable fragment. Note that $\texttt{Cyclist}_{\texttt{SL}}$, `Songbird` and SLS are not complete; for non-valid problems, while $\texttt{Cyclist}_{\texttt{SL}}$ returns `unknown`, `Songbird` and SLS use some heuristic to guess the outcome. For each division, we report the number of correct outputs (`invalid`, `valid`) and the time (in minutes and seconds) taken by each tool. Note that we use the status (`invalid`, `valid`) annotated with each problem in the SL-COMP benchmark as the ground truth. If the output is the same as the status, we classify it as correct; otherwise, it is marked as incorrect. We also note that in these experiments, we used the competition pre-processing tool [39] to transform the SMT 2.6 format into the corresponding formats of the tools before running them. All experiments were performed on an Intel Core i7-6700 CPU 3.4Gh and 8GB RAM. The CPU timeout is 600 seconds.

Experiment results The experimental results are reported in Table 1. In this table, the first column presents the names of the tools. The following three columns show the results of the first division, including the number of correct `invalid` outputs, the number of correct `valid` outputs and the taken time (where *m* for minutes and *s* for seconds), respectively. The number between each pair of brackets *(...)* in the third row shows the number of problems in the corresponding column. Similarly, the following two groups of six columns describe the results of the second and third divisions, respectively.

In general, the experimental results show that $\texttt{S2S}_{\texttt{Lin}}$ is the one (and only one) that could produce all the correct results. Other solvers either produced wrong results or could discharge a fraction of the experiments. Moreover, $\texttt{S2S}_{\texttt{Lin}}$ took a short time for the experiments (8.38 seconds compared to 15.91 seconds for `Spen`, 324 minutes for `Songbird`, 635 minutes for Harrsh, 739 minutes for SLS and 2120 minutes for $\texttt{Cyclist}_{\texttt{SL}}$). While SLS returned 14 false negatives, `Spen` reported 20 false positives. $\texttt{Cyclist}_{\texttt{SL}}$, `Songbird` and Harrsh did not produce any wrong results. Of 569 tests, $\texttt{Cyclist}_{\texttt{SL}}$ could handle 85 tests (15%), Harrsh could handle 215 tests (38%), and `Songbird` could decide on 235 tests (41.3%). In the total of 223 `valid` tests, $\texttt{Cyclist}_{\texttt{SL}}$ could handle 85 problems (38%), and `Songbird` could decide 222 problems (99.5%).

Now we examine the results for each division in detail. For *qf_shls_entl*, `Spen` returned all correct, `Songbird` 186, Harrsh 155, and $\texttt{Cyclist}_{\texttt{SL}}$ 58. If we set the timeout to 2400 seconds, both `Songbird` and Harrsh produced all the correct results. Division

qf_shlid_entl includes 24 `invalid` problems and 36 `valid` problems. While `Songbird` produced 37 problems correctly, $\texttt{Cyclist}_{\texttt{SL}}$ produced 24 correct results. `Spen` reported 27 correct results and 13 false positives (`sll2−vc{01 − 04}` `sll3−vc01`, `sll3−vc{03 − 10}`). The last division, *qf_shlid2_entl*, includes 14 `invalid` and 13 `valid` test problems. While `Songbird` decided only 12 problems correctly, $\texttt{Cyclist}_{\texttt{SL}}$ produced 3 correct outcomes. `Spen` reported 10 correct results. However, it produced 7 false positives (`ls−mul−vc{01 − 03}`, `ls−mul−vc05`, `nll−mul−vc{01 − 03}`). We believe that engineering design and effort play an essential role alongside theory development. Since our experiments provide breakdown results of the two SL-COMP competition divisions, we hope that they provide an initial understanding of the SL-COMP benchmarks and tools. Consequently, this might reduce the effort to prepare experiments over these benchmarks to evaluate new SL solvers. Finally, one might point out that $\texttt{S2S}_{\texttt{Lin}}$ performed well because the entailments in the experiments are within its scope. We do not entirely disagree with this argument but would like to emphasize that tools do not always work well on favourable benchmarks. For example, `Spen` introduced wrong results on *qf_shlid_entl*, and Harrsh did not handle *qf_shlid_entl* and *qf_shlid2_entl* well, although these problems are in their decidable fragments.

7 Related Work

$\texttt{S2S}_{\texttt{Lin}}$ is a variant of the cyclic proof systems [3,4,5,26] and [42]. Unlike existing cyclic proof systems, the soundness of $\texttt{S2S}_{\texttt{Lin}}$ is local, and the proof search is not backtracking. The work presented in [42] shows the completeness of the cyclic proof system. Its main contribution is introducing the rule $*$ for those entailments with a disjunction in the RHS obtained from predicate unfolding. In contrast to [42], our work includes normalization to soundly and completely avoid disjunction in the RHS during unfolding. Moreover, our decidable fragment SHLIDe is non-overlapping to the cone predicates introduced in [42]. Furthermore, due to the empty heap in the base cases, the matching rule in [42] cannot be applied to the predicates in `SHLIDe`. Finally, our work also presents how to obtain the global soundness condition for cyclic proofs.

Our work relates to the inductive theorem provers introduced in [10], [40] and Smallfoot [2]. While [10] is based on structural induction, [40] is based on mathematical induction. Smallfoot [2] proposed a decision procedure for linked lists and trees. It used a fixed compositional rule as a consequence of induction reasoning to handle inductive entailments. Compared with Smallfoot, our proof system replaces the compositional rule by combining rule `LInd` and the back-link construction. Our system could support induction reasoning on a much more expressive fragment of inductive predicates.

Our proposal also relates to works that use lemmas as consequences of induction reasoning [2,16,30,41]. These works in [16,25,30,41] automatically generate lemmas for some classes of inductive predicates. S2 [25] generated lemmas to normalize (such as split and equivalence) the shapes of the synthesized data structures. [16] proposed to generate several sets of lemmas not only for compositional predicates but also for different predicates (e.g., completion lemmas, stronger lemmas and static parameter contraction lemmas). SLS [41] aims to infer general lemmas to prove an entailment. Similarly, S2ENT [30] solves a more generic problem, frame inference, using cyclic

proofs and lemma synthesis. It infers a shape-based residual frame in the LHS and then synthesizes the pure constraints over the two sides.

S2S$_{\text{Lin}}$ relates to model-based decision procedures that reduce the entailment problem in separation logic to a well-studied problem in other domains. For instance, in [8,11,17], the entailment problem, including singly-linked lists and their invariants, is reduced to the problem of inclusion checking in a graph theory. The authors in [18] reduced the entailment problem to the satisfiability problem in second-order monadic logic. This reduction could handle an expressive fragment of spatial-based predicates called bounded-tree width. Moreover, the work presented in [23] shows a model-based decision procedure for a subfragment of the bounded-tree width. Furthermore, while the work in [15,19] reduced the entailment problem to the inclusion checking problem in tree automata, [21] presented an idea to reduce the problem to the inclusion checking problem in heap automata. Moreover, while the procedure in [15] supported compositional predicates (single and double links) well, the procedure in [19] could handle predicates satisfying local properties (e.g., trees with parent pointers). Our decidable fragment subsumes the one described in [2,11,15] but is incomparable to the ones presented in [8,17,18,19]. Works in [34] and [35,36] reduced the entailment problem in separation logic into the satisfiability problem in SMT. While GRASShoper [35,36] could handle transitive closure pure properties, S2S$_{\text{Lin}}$ is capable of supporting local ones. Unlike GRASShoper, which reduces entailment into SMT problems, S2S$_{\text{Lin}}$ reduces an entailment to admissible entailments and detects repetitions via cyclic proofs.

Decidable fragments and complexity results of the entailment problem in separation logic with inductive predicates were well studied. The entailment is 2-EXPTIME in cone predicates [42], the bounded tree-width predicates and beyond [18,14], and EXPTIME in a sub-fragment of cone predicates [19]. In the other class, entailment is in polynomial time for singly-linked lists [11] and semantically linear inductive predicates [15]. Moreover, the extensions with arithmetic [17] are in polynomial but become EXPTIME when the lists are extended with double links [8]. SHLIDe (with nested lists, trees and arithmetic properties) is roughly in the "middle" of the two classes above. The entailment is EXPTIME and becomes polynomial under the upper bound restriction.

8 Conclusion

We have presented a novel decision procedure for the quantifier-free entailment problem in separation logic combined with inductive definitions of compositional predicates and pure properties. Our proposal is the first complete cyclic proof system for the problem in separation logic without back-tracking. We have implemented the proposal in S2S$_{\text{Lin}}$ and evaluated it over the set of nontrivial entailments taken from the SL-COMP competition. The experimental results show that our proposal is effective and efficient when compared to the state-of-the-art solvers. For future work, we plan to develop a bi-abductive procedure based on an extension of this work with the cyclic frame inference procedure presented in [30]. This extension is fundamental to obtaining a compositional shape analysis beyond the lists and trees. Another work is to formally prove that our system is as strong as Smallfoot in the decidable fragment with lists and trees [2]: Given an entailment, if Smallfoot can produce proof, so is S2S$_{\text{Lin}}$.

References

1. Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures*, pages 411–425, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
2. J. Berdine, C. Calcagno, and P. W. O'Hearn. Symbolic Execution with Separation Logic. In *APLAS*, volume 3780, pages 52–68, November 2005.
3. J. Brotherston. Cyclic proofs for first-order logic with inductive definitions. In *Proceedings of TABLEAUX-14*, volume 3702 of *LNAI*, pages 78–92. Springer-Verlag, 2005.
4. J. Brotherston, N. Gorogiannis, and R. L. Petersen. A generic cyclic theorem prover. In *Proceedings of APLAS-10*, LNCS, pages 350–367. Springer, 2012.
5. James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. Automated cyclic entailment proofs in separation logic. In *Proceedings of the 23rd International Conference on Automated Deduction*, CADE'11, page 131–146, Berlin, Heidelberg, 2011. Springer-Verlag.
6. Cristiano Calcagno, Dino Distefano, Jeremy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, pages 3–11, Cham, 2015. Springer International Publishing.
7. Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. In *POPL*, pages 289–300, 2009.
8. Taolue Chen, Fu Song, and Zhilin Wu. Tractability of Separation Logic with Inductive Definitions: Beyond Lists. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:17, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
9. W.-N. Chin, C. Gherghina, R. Voicu, Q.-L. Le, F. Craciun, and S. Qin. A specialization calculus for pruning disjunctive predicates to support verification. In *CAV*. 2011.
10. Duc-Hiep Chu, Joxan Jaffar, and Minh-Thai Trinh. Automatic induction proofs of data-structures in imperative programs. In *Proceedings of PLDI*, PLDI '15, pages 457–466, New York, NY, USA, 2015. ACM.
11. B. Cook, C. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. Tractable reasoning in a fragment of separation logic. In *CONCUR*, volume 6901, pages 235–249. 2011.
12. Christopher Curry, Quang Loc Le, and Shengchao Qin. Bi-abductive inference for shape and ordering properties. In *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 220–225, 2019.
13. Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. Scaling static analyses at facebook. *Commun. ACM*, 62(8):62–70, jul 2019.
14. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Unifying decidable entailments in separation logic with inductive definitions. In *Automated Deduction-CADE 28-28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, pages 183–199, 2021.
15. Constantin Enea, Ondrej Lengál, Mihaela Sighireanu, and Tomás Vojnar. Compositional entailment checking for a fragment of separation logic. *Formal Methods in System Design*, 51(3):575–607, 2017.
16. Constantin Enea, Mihaela Sighireanu, and Zhilin Wu. On automated lemma generation for separation logic with inductive definitions. *ATVA*, 2015.
17. Xincai Gu, Taolue Chen, and Zhilin Wu. *A Complete Decision Procedure for Linearly Compositional Separation Logic with Data Constraints*, pages 532–549. Springer International Publishing, Cham, 2016.

18. R. Iosif, A. Rogalewicz, and J. Simácek. The tree width of separation logic with recursive definitions. In *CADE*, pages 21–38, 2013.
19. Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. *ATVA*, 2014.
20. S. Ishtiaq and P.W. O'Hearn. BI as an assertion language for mutable data structures. In *ACM POPL*, pages 14–26, London, January 2001.
21. Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. *Unified Reasoning About Robustness Properties of Symbolic-Heap Separation Logic*, pages 611–638. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017.
22. Katelaan Jens, Jovanovic Dejan, and Weissenbacher Georg. A separation logic with data: Small models and automation. In *IJCAI*, 2018.
23. Jens Katelaan, Christoph Matheja, and Florian Zuleger. Effective entailment checking for separation logic with inductive definitions. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 319–336, Cham, 2019. Springer International Publishing.
24. Quang Loc Le. Compositional satisfiability solving in separation logic. In Fritz Henglein, Sharon Shoham, and Yakir Vizel, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 578–602, Cham, 2021. Springer International Publishing.
25. Quang Loc Le, Cristian Gherghina, Shengchao Qin, and Wei-Ngan Chin. Shape analysis via second-order bi-abduction. In *CAV*, volume 8559, pages 52–68. 2014.
26. Quang Loc Le and Mengda He. A decision procedure for string logic with quadratic equations, regular expressions and length constraints. In Sukyoung Ryu, editor, *Programming Languages and Systems*, pages 350–372, Cham, 2018. Springer International Publishing.
27. Quang Loc Le and Xuan-Bach D. Le. Artifact for an efficient cyclic entailment procedure in a fragment of separation logic, February 2023. https://doi.org/10.5281/zenodo.7619870.
28. Quang Loc Le and Xuan-Bach D. Le. An efficient cyclic entailment procedure in a fragment of separation logic, January 2023. Technical Report.
29. Quang Loc Le, Jun Sun, and Wei-Ngan Chin. Satisfiability modulo heap-based programs. In *CAV*. 2016.
30. Quang Loc Le, Jun Sun, and Shengchao Qin. Frame inference for inductive entailment proofs in separation logic. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 41–60, 2018.
31. Quang Loc Le, Makoto Tatsuta, Jun Sun, and Wei-Ngan Chin. A decidable fragment in separation logic with inductive predicates and arithmetic. In *CAV*, pages 495–517, 2017.
32. Scott McPeak and George C. Necula. Data structure specifications via local equality axioms. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification*, pages 476–490, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
33. Juan Antonio Navarro Pérez and Andrey Rybalchenko. Separation logic + superposition calculus = heap theorem prover. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, page 556–566, New York, NY, USA, 2011. Association for Computing Machinery.
34. JuanAntonio Navarro Pérez and Andrey Rybalchenko. Separation logic modulo theories. In *APLAS*, volume 8301, pages 90–106. 2013.
35. R. Piskac, T. Wies, and D. Zufferey. Automating separation logic using smt. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044, pages 773–789. 2013.
36. Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic with trees and data. In *CAV*, volume 8559, pages 711–728. 2014.
37. J. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *IEEE LICS*, pages 55–74, 2002.

38. Mihaela Sighireanu and Quang Loc Le. SL-COMP 2022. https://sl-comp.github.io/, 2022. [Online; accessed Jun-2022].
39. Mihaela Sighireanu, Juan Antonio Navarro Pérez, Andrey Rybalchenko, Nikos Gorogiannis, Radu Iosif, Andrew Reynolds, Cristina Serban, Jens Katelaan, Christoph Matheja, Thomas Noll, Florian Zuleger, Wei-Ngan Chin, Quang Loc Le, Quang-Trung Ta, Ton-Chanh Le, Thanh-Toan Nguyen, Siau-Cheng Khoo, Michal Cyprian, Adam Rogalewicz, Tomás Vojnar, Constantin Enea, Ondrej Lengál, Chong Gao, and Zhilin Wu. SL-COMP: competition of solvers for separation logic. In *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics*, pages 116–132, 2019.
40. Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. Automated mutual explicit induction proof in separation logic. In John Fitzgerald, Constance Heitmeyer, Stefania Gnesi, and Anna Philippou, editors, *FM 2016: Proceedings*, pages 659–676, 2016.
41. Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. Automated lemma synthesis in symbolic-heap separation logic. *POPL*, 2018.
42. Makoto Tatsuta, Koji Nakazawa, and Daisuke Kimura. Completeness of cyclic proofs for symbolic heaps with inductive definitions. In Anthony Widjaja Lin, editor, *Programming Languages and Systems*, pages 367–387, Cham, 2019. Springer International Publishing.

Just Testing

Rob van Glabbeek[1,2](✉) ⋆

[1] School of Informatics, University of Edinburgh, Edinburgh, UK
[2] School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
rvg@cs.stanford.edu

Abstract. The concept of must testing is naturally parametrised with a chosen completeness criterion, defining the complete runs of a system. Here I employ justness as this completeness criterion, instead of the traditional choice of progress. The resulting must-testing preorder is incomparable with the default one, and can be characterised as the fair failure preorder of Vogler. It also is the coarsest precongruence preserving linear time properties when assuming justness.

As my system model I here employ Petri nets with read arcs. Through their Petri net semantics, this work applies equally well to process algebras. I provide a Petri net semantics for a standard process algebra extended with signals; the read arcs are necessary to capture those signals.

1 Introduction

May- and must-testing was proposed by De Nicola & Hennessy in [9]. It yields semantic equivalences where two processes are distinguished if and only if they react differently on certain tests. The tests are processes that additionally feature success states. A test $\mathcal{T}$ is applied to a process N by taking the CCS parallel composition $\mathcal{T}|N$, and implicitly applying a CCS restriction operator to it that removes the remnants of unsuccessful communication. Applying $\mathcal{T}$ to N is deemed successful if and only if this composition yields a process that may, respectively must, reach a success state. It is trivial to recast this definition using the CSP parallel composition $\|_{\mathcal{A}}$ [39] instead of the one from CCS.

It is not a priori clear how a given process *must* reach a success state. For all we know it might stay in its initial state and never take any transition leading to this success state. To this end one must employ an assumption saying that under appropriate circumstances certain enabled transitions will indeed be taken. Such an assumption is called a *completeness criterion* [18]. The theory of testing from [9] implicitly employs a default completeness criterion that in [25] is called *progress*. However, one can parameterise the notion of must testing by the choice of any completeness criterion, such as the many notions of *fairness* classified in [25]. Here I employ *justness*, a completeness criterion that is better justified than either progress or fairness [25].

⋆ Supported by Royal Society Wolfson Fellowship RSWF\R1\221008

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 498–519, 2023.
https://doi.org/10.1007/978-3-031-30829-1_24

The resulting must-testing equivalence is incomparable to the progress-based one from [9]. On the one hand, it no longer distinguishes deadlock and livelock, i.e., the Petri nets N and N' of Ex. 3; on the other hand, it keeps recording information past a divergence. I characterise the corresponding preorder as the fair failure preorder of Vogler [43], which using my terminology ought to be called the *just failures preorder*. I show that it also is the coarsest precongruence preserving linear time properties when assuming justness. Finally I show that the same preorder originates from the timed must-testing framework explored in [43], but only if all quantitative information is removed from that approach.

I carry out this work within the model of Petri nets extended with read arcs [35,7], so that it also applies to process algebras through their standard Petri net semantics. The extension with read arcs is necessary to capture *signalling*, a process algebra operator that cannot be adequately modelled by standard Petri nets. Signalling, or read arcs, can be used to accurately model mutual exclusion without making a fairness assumption [43,8,11]. This is not possible in standard Petri nets [31,43,24], or in process algebras with a standard Petri net semantics [24]. Here I give a Petri net semantics of signalling, and illustrate its use in modelling a traffic light, interacting with passing cars.

Acknowledgement I am grateful to Weiyou Wang for valuable feedback.

2 Labelled Petri nets with read arcs

I will employ the following notations for multisets.

Definition 1 Let X be a set.

- A *multiset* over X is a function $A\colon X \to \mathbb{N}$, i.e. $A \in \mathbb{N}^X$.
- $x \in X$ is an *element of* A, notation $x \in A$, iff $A(x) > 0$.
- For multisets A and B over X I write $A \subseteq B$ iff $A(x) \leq B(x)$ for all $x \in X$;
  $A \cup B$ denotes the multiset over X with $(A \cup B)(x) := \max(A(x), B(x))$,
  $A \cap B$ denotes the multiset over X with $(A \cap B)(x) := \min(A(x), B(x))$,
  $A + B$ denotes the multiset over X with $(A + B)(x) := A(x) + B(x)$,
  $A - B$ is given by $(A - B)(x) := \max(A(x) - B(x), 0)$, and
  for $k \in \mathbb{N}$ the multiset $k \cdot A$ is given by $(k \cdot A)(x) := k \cdot A(x)$.
- The function $\emptyset : X \to \mathbb{N}$, given by $\emptyset(x) := 0$ for all $x \in X$, is the *empty* multiset over X.
- The cardinality $|A|$ of a multiset A over X is given by $|A| := \sum_{x \in X} A(x)$.
- A multiset A over X is *finite* iff $|A| < \infty$, i.e., iff the set $\{x \mid x \in A\}$ is finite.

With $\{x, x, y\}$ I denote the multiset over $\{x, y\}$ with $A(x)$=2 and $A(y)$=1, rather than the set $\{x, y\}$ itself. A multiset A with $A(x) \leq 1$ for all x is identified with the set $\{x \mid A(x) = 1\}$.

I employ general labelled place/transition systems extended with read arcs [35,7].

Definition 2 Let $\mathcal{A}$ be a set of *visible actions* and $\tau \notin \mathcal{A}$ be an *invisible action*. Let $\mathcal{A}_\tau := \mathcal{A} \dot\cup \{\tau\}$. A (*labelled*) *Petri net* (*over* $\mathcal{A}_\tau$) is a tuple (S, T, F, R, M_0, ℓ) where
- S and T are disjoint sets (of *places* and *transitions*),
- $F : ((S \times T) \cup (T \times S)) \to \mathbb{N}$ (the *flow relation* including *arc weights*),
- $R : S \times T \to \mathbb{N}$ (the *read* relation),
- $M_0 : S \to \mathbb{N}$ (the *initial marking*), and
- $\ell : T \to \mathcal{A}_\tau$ (the *labelling function*).

Petri nets are depicted by drawing the places as circles and the transitions as boxes, containing their label. Identities of places and transitions are displayed next to the net element. When $F(x, y) > 0$ for $x, y \in S \cup T$ there is an arrow (*arc*) from x to y, labelled with the *arc weight* $F(x, y)$. Weights 1 are elided. An element (s, t) of the multiset R is called a *read arc*. Read arcs are drawn as lines without arrowhead. When a Petri net represents a concurrent system, a global state of this system is given as a *marking*, a multiset M of places, depicted by placing $M(s)$ dots (*tokens*) in each place s. The initial state is M_0.

The behaviour of a Petri net is defined by the possible moves between markings M and M', which take place when a finite multiset G of transitions *fires*. In that case, each occurrence of a transition t in G consumes $F(s, t)$ tokens from each place s. Naturally, this can happen only if M makes all these tokens available in the first place. Moreover, for each $t \in G$ there need to be at least $R(s, t)$ tokens in each place s that are not consumed when firing G. Next, each t produces $F(t, s)$ tokens in each place s. Definition 4 formalises this notion of behaviour.

Definition 3 Let $N = (S, T, F, R, M_0, \ell)$ be a Petri net. The multisets $\widehat{t}$, ${}^\bullet t$, $t^\bullet : S \to \mathbb{N}$ are given by $\widehat{t}(s) = R(s, t)$, ${}^\bullet t(s) = F(s, t)$ and $t^\bullet(s) = F(t, s)$ for all $s \in S$. The elements of $\widehat{t}$, ${}^\bullet t$ and $t^\bullet$ are called *read-*, *pre-* and *postplaces* of t, respectively. These functions extend to finite multisets G: $T \to \mathbb{N}$ by $\widehat{G} := \bigcup_{t \in G} \widehat{t}$, ${}^\bullet G := \sum_{t \in T} G(t) \cdot {}^\bullet t$ and $G^\bullet := \sum_{t \in T} G(t) \cdot t^\bullet$.

Definition 4 ([7]) Let $N = (S, T, F, R, M_0, \ell)$ be a Petri net, $G \in \mathbb{N}^T$ non-empty and finite, and $M, M' \in \mathbb{N}^S$. G is a *step* from M to M', written $M\ [G\rangle_N\ M'$, iff
- ${}^\bullet G + \widehat{G} \subseteq M$ (G is *enabled*) and
- $M' = (M - {}^\bullet G) + G^\bullet$.

Note that steps are (finite) multisets, thus allowing self-concurrency, i.e. the same transition can occur multiple times in a single step. One writes $M\ [t\rangle_N\ M'$ for $M\ [\{t\}\rangle_N\ M'$, whereas $M[t\rangle_N$ abbreviates $\exists M'.\ M\ [t\rangle_N\ M'$. The subscript N may be omitted if clear from context.

In my Petri nets transitions are labelled with *actions* drawn from a set $\mathcal{A} \dot\cup \{\tau\}$. This makes it possible to see these nets as models of *reactive systems* that interact with their environment. A transition t can be thought of as the occurrence of the action $\ell(t)$. If $\ell(t) \in \mathcal{A}$, this occurrence can be observed and influenced by the environment, but if $\ell(t) = \tau$, it cannot and t is an *internal* or *silent* transition. Transitions whose occurrences cannot be distinguished by the

environment carry the same label. In particular, since the environment cannot observe the occurrence of internal transitions at all, they are all labelled τ.

In [31,43,24] it was established that mutual exclusion protocols cannot be correctly modelled in standard Petri nets (without read arcs, i.e., satisfying $R(s,t) = 0$ for all $s \in S$ and $t \in T$), unless their correctness becomes contingent on making a fairness assumption. In [24] it was concluded from this that mutual exclusion protocols can likewise not be correctly expressed in standard process algebras such as CCS [34], CSP [6] or ACP [4], at least when sticking to their standard Petri net semantics. Yet Vogler showed that mutual exclusion can be correctly modelled in Petri nets with read arcs [43], and [8,11] demonstrate how mutual exclusion can be correctly modelled in a process algebra extended with *signalling* [3]. Thus signalling adds expressiveness to process algebra that cannot be adequately modelled in terms of standard Petri nets. This is my main reason to use Petri nets with read arcs as system model in this paper.

In many papers on Petri nets, the sets of places and transitions are required to be finite, or at least countable. Here I need a milder restriction, and will limit attention to nets that are finitary in the following sense.

Definition 5 A Petri net $N = (S, T, F, R, M_0, \ell)$ is *finitary* if M_0 is countable, $t^\bullet$ is countable for all $t \in T$, and moreover the set of transitions t with ${}^\bullet t = \emptyset$ is countable.

3 A Petri net semantics of CCSP with signalling

CCSP [37] is a natural mix of the process algebras CCS [34] and CSP [6], often used in connection with Petri nets. Here I will present a Petri net semantics of a version CCSPS of CCSP enriched with *signalling* [3]. This builds on work from [29,44,27,10,37,38]; the only novelty is the treatment of signalling. Petri net semantics of other process algebras, like CCS [34], CSP [6] or ACP [4], are equally well known. This Petri net semantics lifts any semantic equivalence on Petri nets to CCSPS, or to any other process algebra, so that the results of this work apply equally well to process algebras.

CCSPS is parametrised by the choice of sets $\mathcal{A}$ of visible actions and $\mathcal{K}$ of *agent identifiers*. Its syntax is given by

$$P, Q, P_i ::= \sum_{i \in I} a_i P_i \;\mid\; a \triangleright \sum_{i \in I} a_i P_i \;\mid\; P \|_A Q \;\mid\; \tau_A(P) \;\mid\; f(P) \;\mid\; K$$

with $a, a_i \in \mathcal{A}$, $A \subseteq \mathcal{A}$, $f : \mathcal{A} \to \mathcal{A}$ and $K \in \mathcal{K}$. Here the guarded choice $\sum_{i \in I} a_i P_i$ executes one of the actions a_i, followed by the process P_i. The process $a \triangleright P$ behaves as P, except that in its initial state it it is sending the signal a.[1] [2] The process $P \|_A Q$ is the partially synchronous parallel composition of processes

[1] The notation $a \triangleright P$ follows [8]; in [3,11] this is denoted $P \hat{} a$.

[2] Here I require P to be a guarded choice in order to avoid the need for a *root condition* [13] to make the equivalences of this paper into congruences. This is also the reason my language features a guarded choice, instead of action prefixing and general choice.

P and Q, where actions from A can take place only when both P and Q can engage in such an action, while other actions of P and Q occur independently. The abstraction operator τ_A hides action from A from the environment by renaming them into τ, whereas f is a straightforward relabelling operator (leaving internal actions alone). Each agent identifier K comes with a *defining equation* $K \stackrel{def}{=} P$, with P a *guarded* CCSPS expression; it behaves exactly as the body of its defining equation. Here P is guarded if each occurrence of an agent identifier within P lays in the scope of a guarded choice $\sum_{i\in I} a_i P_i$ or $a \triangleright \sum_{i\in I} a_i P_i$.

A formal Petri net semantics of CCSPS, and of each of the operators $\sum$, $\triangleright$, $\|_A$, τ_A and f, appears in [22, Appendix A]. Here I give an informal summary.

Given nets N_i for $i \in I$, the net $\sum_{i\in I} a_i N_i$ is obtained by taking their disjoint union, but without their initial markings $(M_0)_i$, and adding a single marked place r, and for each $i \in I$ a fresh transition t_i, labelled a_i, with ${}^\bullet t_i = \{r\}$, $\widehat{t_i} = \emptyset$ and $({}^\bullet t_i) = (M_0)_i$.

The parallel composition $N\|_A N'$ is obtained out of the disjoint union of N and N' by dropping from N and N' all transitions t with $\ell(t) \in A$, and instead adding synchronisation transitions (t, t') for each pair of transitions t and t' from N and N' with $\ell(t) = \ell(t') \in A$. One has ${}^\bullet(t, t') := {}^\bullet t + {}^\bullet t'$, and similarly for $\widehat{(t, t')}$ and $(t, t')^\bullet$, i.e., all arcs are inherited.

τ_A and f are renaming operators that only affect the labels of transitions.

The net $a \triangleright N$ adds to the net N a single transition u, labelled a, that may fire arbitrary often, but is enabled in the initial state of N only. To this end, take ${}^\bullet u = u^\bullet = \emptyset$ and $\widehat{u} = M_0$, the initial marking of N. I apply this construction only to nets for which its initially marked places have no incoming arcs.

Example 1 A traffic light can be modelled by the recursive equation

$$TL \stackrel{def}{=} tr.tg.(drive \triangleright ty.TL).$$

Here the actions *tr*, *tg* and *ty* stand for "turn red", "turn green" and "turn yellow", and *drive* indicates a state where it is OK to drive through. A sequence of two passing cars is modelled as $Traffic \stackrel{def}{=} drive.drive.\mathbf{0}$. Here $\mathbf{0}$ stands for the empty sum $\sum_{i\in\emptyset} a_i.E_i$ and models inaction. In the parallel composition $TL \|_{\{drive\}} Traffic$ the cars only drive through when the light is green. All three processes are displayed in Fig. 1.

4 Justness and other completeness criteria

Definition 6 Let $N = (S, T, F, R, M_0, \ell)$ be a Petri net. An *execution path* π is an alternating sequence $M_0 t_1 M_1 t_2 M_2 \ldots$ of markings and transitions of N, starting with M_0, and either being infinite or ending with a marking, such that $M_i\ [t_{i+1}\rangle_N\ M_{i+1}$ for all $i < length(\pi)$. Here $length(\pi) \in \mathbb{N} \cup \{\infty\}$ is the number of transitions in π.

Let $\ell(\pi) \in \mathcal{A}_\tau^\infty$ be the string $\ell(t_1)\ell(t_2)\ldots$. Here $\mathcal{A}_\tau^\infty$ denotes the collection of finite and infinite sequences of actions. Moreover, $trace(\pi) \in \mathcal{A}^\infty$ is obtained from $\ell(\pi)$ by dropping all occurrences of τ.

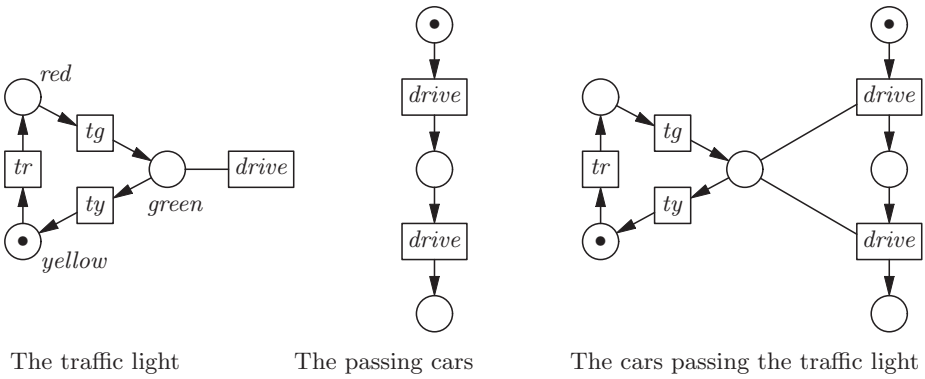


Fig. 1. Traffic passing traffic light

The execution path π is said to *enable* a transition t, notation $\pi[t\rangle$, if $M_k[t\rangle$ for some $k \in \mathbb{N} \wedge k \leq \mathit{length}(\pi)$ and for all $k \leq j < \mathit{length}(\pi)$ one has $t_j \neq t$ and $({}^\bullet t + \widehat{t}\,) \cap {}^\bullet t_{j+1} = \emptyset$.

Path π is *B-just*, for some $B \subseteq \mathcal{A}$, if $\ell(t) \in B$ for all $t \in T$ with $\pi[t\rangle$.

In the definition of $\pi[t\rangle$ above one also has $M_{j+1}[t\rangle$ for all $k \leq j < \mathit{length}(\pi)$. Hence, a finite execution path enables a transition iff its final marking does so.

Informally, $\pi[t\rangle$ holds iff transition t is enabled in some marking on the path π, and after that state no transition of π uses any of the resources needed to fire t. Here the read- and preplaces of t count as such resources. The clause $t_j \neq t$ moreover counts the transition itself as one of its resources, in the sense that a transition is no longer enabled when it occurs. This clause is redundant for transitions t with ${}^\bullet t \neq \emptyset$. One could interpret this clause as saying that a transition t with ${}^\bullet t = \emptyset$ comes with implicit marked private preplace p_t, and arcs (p_t, t) as well as (t, p_t).

In [18] I posed that Petri nets or transition systems constitute a good model of concurrency only in combination with a *completeness criterion*: a selection of a subset of all execution paths as complete executions, modelling complete runs of the represented system. The default completeness criterion, called *progress* in [25], declares an execution path complete iff it either is infinite, or its final marking enables no transition. An alternative, called *justness* in [25], declares an execution path complete iff it enables no transition. Justness is a *stronger* completeness criterion than progress, in the sense that it deems fewer execution paths complete. The difference is illustrated by the Petri net of Fig. 2(a). There, the execution of an infinite sequence of b-transitions, not involving the a-transition,

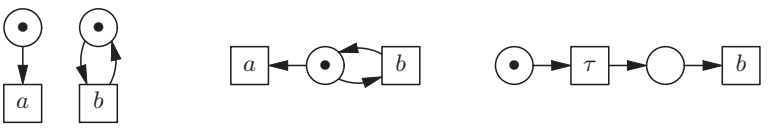


Fig. 2. (a) Progress vs. justness; (b) Justness vs. fairness; (c) $\{b\}$-progress vs. $\emptyset$-progress

is complete when assuming progress, but not when assuming justness. In the survey paper [25], 20 different completeness criteria are ordered by strength: progress, justness, and 18 kinds of fairness. Most of the latter are stronger than justness: in Fig. 2(b) the infinite sequence of b-transitions is just but unfair—i.e. incomplete according to these notions of fairness. Whereas justness was a new idea in the context of transition systems [25], it was used as an unnamed default assumption in much work on Petri nets [40]. That justness is better warranted in applications than other completeness criteria has been argued in [25,18,24,17].

The mentioned completeness criteria from [25] are all stronger than progress, in the sense that not all infinite execution paths are deemed complete; on the finite execution paths they judge the same. An orthogonal classification is obtained by varying the set $B \subseteq \mathcal{A}$ of actions that may be blocked by the environment. This fits the reactive viewpoint, in which a visible action can be regarded as a synchronisation between the modelled system and its environment. An environment that is not ready to synchronise with an action $b \in \mathcal{A}$ can be regarded as blocking b. Now *B-progress* is the criterion that deems a path complete iff it is either infinite, or its final marking M enables only transitions with labels from B. When the environment may block such transitions, it is possible for the system to not progress past M. In Fig. 2(c) the execution that performs only the τ-transition is complete when assuming $\{b\}$-progress, but not when assuming $\emptyset$-progress. Definition 6 defines B-justness accordingly, and [25] furthermore defines 18 different notions of B-fairness, for any choice of $B \subseteq \mathcal{A}$. The internal action $\tau \notin B$ can never be blocked by the environment. The default forms of progress and justness described above correspond with $\emptyset$-progress and $\emptyset$-justness. In [40] blocking and non-blocking transitions are called *cold* and *hot*, respectively.

Two subtly different computational interpretations of Petri nets appear in the literature [14]: in the *individual token interpretation* multiple tokens appearing in the same place are seen as different resources, whereas in the *collective token interpretation* only the number of tokens in a place is semantically relevant. The difference is illustrated in Fig. 3.

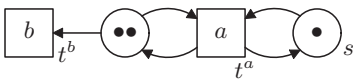


Fig. 3. Run a^∞ is just under the individual token interpretation of Petri nets

The idea underlying justness is that once a transition t is enabled, eventually either t will fire, or one of the resources necessary for firing t will be used by some other transition. The execution path π in the net of Fig. 3 that fires the action a infinitely often, but never the action b, is $\emptyset$-just by Def. 6. Namely, t^b is not enabled by π, as $({}^\bullet t^b + \widehat{t^b}) \cap {}^\bullet t^a \neq \emptyset$. This fits with the individual token interpretation, as in this run it is possible to eventually consume each token that is initially present, and each token that stems from firing transition t^a. This way any resource available for firing t^b will eventually be used by some other transition.

When adhering to the collective token interpretation of nets, execution path π could be deemed $\emptyset$-unjust, since transition t^b can fire when there is at least one token in its preplace, and this state of affairs can be seen as a single resource that is never taken away. This might be formalised by adapting the definition of $\pi[t\rangle$, a path enabling a transition, namely by changing the condition $({}^\bullet t + \widehat{t}) \cap {}^\bullet t_{j+1} = \emptyset$ from Def. 6 into ${}^\bullet t + \widehat{t} + {}^\bullet t_{j+1} \subseteq M_j$. However, this formalisation doesn't capture that after dropping place s from the net of Fig. 3 there is still an infinite run in which b does not occur, namely when regularly firing two as simultaneously. This contradicts the conventional wisdom that firing multiple transitions at once can always be reduced to firing them in some order. To avoid that type of complication, I here stick to the individual token interpretation. Alternatively, one could restrict attention to 1-safe nets [40], on which there is no difference between the individual and collective token interpretations, or to the larger class of *structural conflict nets* [23,21], on which the conditions $({}^\bullet t + \widehat{t}) \cap {}^\bullet t_{j+1} = \emptyset$ and ${}^\bullet t + \widehat{t} + {}^\bullet t_{j+1} \subseteq M_j$ are equivalent [21, Section 23.1], so that Def. 6 applies equally well to the collective token interpretation.

5 Feasibility

A standard requirement on fairness assumptions, or completeness criteria in general, is *feasibility* [2], called *machine closure* in [33]. It says that any finite execution path can be extended into a complete one. The following theorem shows that B-justness is feasible indeed.

Theorem 1 For any $B \subseteq \mathcal{A}$, each finite execution path of a finitary Petri net can be extended into a B-just path.

Proof. Without loss of generality I restrict attention to nets without transitions t with ${}^\bullet t = \emptyset$. Namely, an arbitrary net can be enriched with marked private preplaces p_t for each such t, and arcs (p_t, t) and (t, p_t). In essence, this enrichment preserves the collection of execution path of the net, ordered by the relation "is an extension of", the validity of statements $\pi[t\rangle$, and the property of B-justness.

I present an algorithm extending any given path $M_0 t_1 M_1 t_2 \dots t_{k-1} M_k$ into a B-just path $\pi = M_0 t_1 M_1 t_2 M_2 \dots$. The extension only uses transitions t_i with $\ell(t_i) \notin B$. As data structure my algorithm employs an $\mathbb{N} \times \mathbb{N}$-matrix with columns named i, for $i \geq k$, where each column has a head and a body. The head of column k contains M_k and its body lists the places $s \in M_k$, leaving empty most slots if there are only finitely many such places. Since the given net is finitary, M_k has only countable many elements, so that they can be listed in the $\mathbb{N}$ slots of column k.

The head of each column $i > k$ with $i{-}1 < \mathit{length}(\pi)$ will contain the pair (t_i, M_i) and its body will list the places $s \in M_i$, again leaving empty most slots if there are only finitely many such places. Once more, finitariness ensures that there are enough slots in column i.

An entry in the body of the matrix is either (still) empty, filled in with a place, or crossed out. Let $f : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ be an enumeration of the entries in the body of this matrix.

At the beginning only column k is filled in; all subsequent columns of the matrix are empty. At each step $i > k$ I first cross out all entries s in the body of the matrix for which there is no transition t with $\ell(t) \notin B$, $M_{i-1}[t\rangle$ and $s \in {}^\bullet t$. In case all entries of the matrix are crossed out, the algorithm terminates, with output $M_0 t_1 M_1 t_2 \dots M_{i-1}$. Otherwise I fill in column i as follows and cross out some more places occurring in body of the matrix.

I take n to be the smallest value such that entry $f(n) \in \mathbb{N} \times \mathbb{N}$ is already filled in, say with place r, but not yet crossed out. By the previous step of the algorithm, $M_{i-1}[t_i\rangle$ for some transition t_i with $\ell(t_i) \notin B$ and $r \in {}^\bullet t_i$. I now fill in (t_i, M_i) in the head of column i; here M_i is the unique marking such that $M_{i-1}[t_i\rangle M_i$. Subsequently I cross out all entries in the body of the matrix containing a place $r' \in {}^\bullet t_i$. This includes the entry $f(n)$. Finally, I fill in the body of column i with the places $s \in M_i$.

In case the algorithm doesn't terminate, the desired path π is the sequence $\pi = M_0 t_1 M_1 t_2 M_2 \dots$ that is constructed in the limit. It remains to show that π is B-just.

Towards a contradiction, suppose $\pi[t\rangle$ for a transition t with $\ell(t) \notin B$. By Def. 6 there is an $m \in \mathbb{N} \wedge m \leq \mathit{length}(\pi)$ such that $M_m[t\rangle$ and $({}^\bullet t + \widehat{t}) \cap {}^\bullet t_{j+1} = \emptyset$ for all $m \leq j < \mathit{length}(\pi)$. Let h be the smallest such m with $m \geq k$. Then there is a place $r \in {}^\bullet t$ appearing in column h. Here I use that ${}^\bullet t \neq \emptyset$. This place was not yet crossed out when column h was constructed. Since $r \notin {}^\bullet t_{j+1}$ and $M_{j+1}[t\rangle$ for all $h \leq j < \mathit{length}(\pi)$, place r will never be crossed out. It follows that π must be infinite. The entry r in column h is enumerated as $f(n)$ for some $n \in \mathbb{N}$, and is eventually reached by the algorithm and crossed out. In this regard the matrix acts as a priority queue. This yields the required contradiction. □

The above proof is a variant of [18, Thm. 1], which itself is a variant of [25, Thm. 6.1]. The side condition of finitariness is essential, as the below counterexample shows.

Example 2 Let $N = (S, T, F, R, M_0, \ell)$ be the net with $T = \{t_r \mid r \in \mathbb{R}\}$, $S = \{s_r \mid r \in \mathbb{R}\}$, $M_0(s_r) = 1$, $\ell(t_r) = \tau$, ${}^\bullet t_r = \{s_r\}$ and $\widehat{t_r} = t_r^\bullet = \emptyset$ for each $r \in \mathbb{R}$. It contains uncountably many action transitions, each with a marked private preplace. As each execution path π contains only countably many transitions, many transitions remain enabled by π.

6 The coarsest preorders preserving linear time properties

A *linear time property* is a predicate on system runs, and thus also on the execution paths of Petri nets. One writes $\pi \models \varphi$ if the execution path π satisfies the linear-time property φ. As the observable behaviour of an execution path π of a Petri net is deemed to be $\mathit{trace}(\pi)$, in this context one studies only linear

time properties φ such that

$$trace(\pi) = trace(\pi') \quad \Leftrightarrow \quad (\pi \models \varphi \Leftrightarrow \pi' \models \varphi)\ . \tag{1}$$

For this reason, a linear time property can be defined or characterised as a subset of $\mathcal{A}^\infty$.

Linear time properties can be used to formalise correctness requirements on systems. They are deemed to hold for (or be satisfied by) a system iff they hold for all its complete runs. Following [20] I write $\mathscr{D} \models^{CC} \varphi$ iff property φ holds for all runs of the distributed system $\mathscr{D}$—and $N \models^{CC} \varphi$ iff it holds for all execution paths of the Petri net N—that are complete according to the completeness criterion CC. Prior to [20], $\models$ was a binary predicate predicate between systems—or system representations such as Petri nets—and properties; in this setting the default completeness criterion of Section 4 was used. When using a completeness criterion B-C, where C is one of the 20 completeness criteria classified in [25] and $B \subseteq \mathcal{A}$ is a modifier of C based on the set B of actions that may be blocked by the environment, $N \models^{B\text{-}C} \varphi$ is written $N \models^C_B \varphi$ [20]. In this paper I am mostly interested in the values Pr and J of C, standing for progress and justness, respectively. To be consistent with previous work on temporal logic, $N \models \varphi$ is a shorthand for $N \models^{Pr}_{\emptyset} \varphi$.

For each completeness criterion B-C, let $\sqsubseteq^C_B$ be the coarsest preorder that preserves linear time properties when assuming B-C. Moreover, $\sqsubseteq^C$ is the coarsest preorder that preserves linear time properties when assuming completeness criterion C in each environment, meaning regardless which set of actions B can be blocked.

Definition 7 Write $N \sqsubseteq^C_B N'$ iff $N \models^C_B \varphi \ \Rightarrow \ N' \models^C_B \varphi$ for all linear time properties φ. Write $N \sqsubseteq^C N'$ iff $N \sqsubseteq^C_B N'$ for all $B \subseteq \mathcal{A}$.

It is trivial to give a more explicit characterisation of these preorders. To preserve the analogy with the failure pairs of CSP [6], instead of sets $B \subseteq \mathcal{A}$ I will record their complements $\overline{B} := \mathcal{A} \backslash B$. As $\overline{\overline{B}} = B$, such sets carry the same information. Since B contains the actions that *may* be blocked by the environment, meaning that we consider environments that in any state may decide which actions from B to block, the set $\overline{B} \cup \{\tau\}$ contains actions that may not be blocked by the environment. This means that we only consider environments that in any state are willing to synchronise with any action in $\overline{B}$.

Definition 8 For completeness criterion C, B ranging over $\mathscr{P}(\mathcal{A})$, and Petri net N, let

$$\mathscr{F}^C(N) := \{(\sigma, \overline{B}) \mid N \text{ has a } B\text{-}C\text{-complete execution path } \pi \text{ with } \sigma = trace(\pi)\}$$
$$\mathscr{F}^C_B(N) := \{\ \sigma \mid N \text{ has a } B\text{-}C\text{-complete execution path } \pi \text{ with } \sigma = trace(\pi)\}.$$

An element (σ, X) of $\mathscr{F}^C(N)$ could be called a *C-failure pair* of N, because it indicates that the system represented by N, when executing a path with visible content σ, may fail to execute additional actions from X, even when all these

actions are offered by the environment, in the sense that the environment is perpetually willing to partake in those actions. Note that if $(\sigma, X) \in \mathscr{F}^C(N)$ and $Y \subseteq X$ then $(\sigma, Y) \in \mathscr{F}^C(N)$.

Proposition 1 $N \sqsubseteq^C_B N'$ iff $\mathscr{F}^C_B(N) \supseteq \mathscr{F}^C_B(N')$.
Likewise, $N \sqsubseteq^C N'$ iff $\mathscr{F}^C(N) \supseteq \mathscr{F}^C(N')$.

Proof. Suppose $N \sqsubseteq^C_B N'$ and $\sigma \notin \mathscr{F}^C_B(N)$. Let φ be the linear time property satisfying $\pi \models \varphi$ iff $\mathit{trace}(\pi) \neq \sigma$. Then $N \models^C_B \varphi$ and thus $N' \models^C_B \varphi$. Hence $\sigma \notin \mathscr{F}^C_B(N')$.

Suppose $N \not\sqsubseteq^C_B N'$. There there exists a linear time property φ such that $N \models^C_B \varphi$, yet $N' \not\models^C_B \varphi$. Let π' be a B-C-complete execution path of N' such that $\pi' \not\models \varphi$, and let $\sigma = \mathit{trace}(\pi')$. By (1) $\pi \not\models \varphi$ for any execution path π (of any net) such that $\mathit{trace}(\pi) = \sigma$. Hence $\sigma \in \mathscr{F}^C_B(N')$, yet $\sigma \notin \mathscr{F}^C_B(N)$. It follows that $\mathscr{F}^C_B(N) \not\supseteq \mathscr{F}^C_B(N')$.

The second statement follows as a corollary of the first, using that $\mathscr{F}^C(N) \supseteq \mathscr{F}^C(N')$ iff $\mathscr{F}^C_B(N) \supseteq \mathscr{F}^C_B(N')$ for all $B \subseteq \mathcal{A}$. □

The preorders $\sqsubseteq^C_B$ can be classified as linear time semantics [12], as they are characterised through reverse trace inclusions. The preorders $\sqsubseteq^C$ on the other hand capture a minimal degree of branching time. This is because they should be ready for different choices of a system's environment at runtime.

Note that $\sqsubseteq^C$ is contained in $\sqsubseteq^C_B$ for each $B \subseteq \mathcal{A}$, in the sense that $N \sqsubseteq^C N'$ implies $N \sqsubseteq^C_B N'$. There is a priori no reason to assume inclusions between preorders $\sqsubseteq^C$ and $\sqsubseteq^D$ when D is a stronger completeness criterion than C.

To relate the preorders $\sqsubseteq^C_B$ and $\sqsubseteq^C$ with ones established in the literature, I consider the case $C = Pr$, i.e., taking progress as the completeness criterion C. The preorder $\sqsubseteq^{Pr}_\emptyset$ is characterised as reverse inclusion of complete traces, where completeness is w.r.t. the default completeness criterion of Section 4. These complete traces include

- the infinite traces of a system,
- its *divergence traces* (stemming from execution paths that end in infinitely many τ-transitions), and
- its *deadlock traces* (stemming from finite execution paths that end in a marking enabling no transitions).

Deadlock and divergence traces are not distinguished. This corresponds with what is called *divergence sensitive trace semantics* (T^λ) in [12]. The above concept of complete traces of a process p is the same as in [15], there denoted $CT(p)$.

The preorder $\sqsubseteq^{Pr}_{\mathcal{A}}$ is characterised as reverse inclusion of infinite and partial traces, i.e., the traces of *all* execution paths. This corresponds with what is called *infinitary trace semantics* (T^∞) in [12]. It is strictly coarser (making more identifications) than T^λ.

To analyse the preorder $\sqsubseteq^{Pr}$, one has $(\sigma, X) \in \mathscr{F}^{Pr}(N)$ if either

- σ is an infinite trace of N—the set X plays no rôle in that case,
- σ is a divergence trace of N, or

– σ is the trace of a finite path of N whose end-marking enables no transition t with $\ell(t) \in X$.

The resulting preorder does not occur in [12]—it can be placed strictly between *divergence sensitive failure semantics* (F^{Δ}) and *divergence sensitive trace semantics* (T^{λ}).

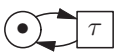
The entire family of preorders $\sqsubseteq_B^C$ and $\sqsubseteq^C$ proposed in this section was inspired by its most interesting family member, $\sqsubseteq^J$ (i.e., taking justness as the completeness criterion C), proposed earlier by Walter Vogler [43, Def. 5.6], also on Petri nets with read arcs. Vogler [43] uses the word *fair* for what I call *just*. I believe the choice of the word "just" is warranted to distinguish the concept from the many other kinds of fairness that appear in the literature, which are all of a very different nature. Accordingly, Vogler calls the semantics induced by $\sqsubseteq^J$ the *fair failure* semantics, whereas I call it the *just failures* semantics. My set $\mathscr{F}^J(N)$ is called $\mathscr{F}\mathscr{F}(N)$ in [43], and Vogler addresses $\sqsupseteq^J$ simply as $\mathscr{F}\mathscr{F}$-inclusion, thereby defining it via the right-hand side of Prop. 1.

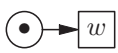
7 Congruence properties

A preorder $\sqsubseteq$ is called a *precongruence* for an n-ary operator Op, if $N_i \sqsubseteq N_i'$ for $i = 1, \ldots, n$ implies that $Op(N_1, \ldots, N_n) \sqsubseteq Op(N_1', \ldots, N_n')$. In this case the operator Op is said to be *monotone* w.r.t. the preorder $\sqsubseteq$. Being a precongruence for important operators is known to be a valuable tool in compositional verification [41].

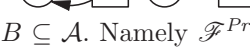
I write $\equiv$ for the kernel of $\sqsubseteq$, that is, $N \equiv N'$ iff $N \sqsubseteq N' \wedge N' \sqsubseteq N$. Here I also imply that $\equiv_B^C$ is the kernel of $\sqsubseteq_B^C$. If $\sqsubseteq$ is a precongruence for Op, then $\equiv$ is a *congruence* for Op, meaning that $N_i \equiv N_i'$ for $i = 1, \ldots, n$ implies that $Op(N_1, \ldots, N_n) \equiv Op(N_1', \ldots, N_n')$.

The preorder $\sqsubseteq_{\mathcal{A}}^{Pr}$, characterised as reverse inclusion of infinite and partial traces, is well-known to be precongruence for the operators of CCSP. However, none of the other preorders $\sqsubseteq_B^{Pr}$, nor $\sqsubseteq^{Pr}$, is a precongruence for parallel composition.

Example 3 Let $N =$  , $N' =$ 

and $\mathcal{T} =$ 

. The definition of $\|_\emptyset$ yields $\mathcal{T}\|_\emptyset N =$ 

and $\mathcal{T}\|_\emptyset N' =$ 

. One has $N \equiv^{Pr} N'$, and thus also $N \equiv_B^{Pr} N'$, for each $B \subseteq \mathcal{A}$. Namely $\mathscr{F}^{Pr}(N) = \mathscr{F}^{Pr}(N') = \{(\varepsilon, X) \mid X \subseteq \mathcal{A}\}$. Here ε denotes the empty string. When fixing B such that $B \neq \mathcal{A}$ one may choose $w \notin B$. Now $\varepsilon \in \mathscr{F}_B^{Pr}(\mathcal{T}\|_\emptyset N')$, for this process has an infinite execution path that avoids the w-transition, which generates a divergence trace ε. Yet $\varepsilon \notin \mathscr{F}_B^{Pr}(\mathcal{T}\|_\emptyset N)$. Hence $\mathcal{T}\|_\emptyset N \not\sqsubseteq_B^{Pr} \mathcal{T}\|_\emptyset N'$, and thus also $\mathcal{T}\|_\emptyset N \not\sqsubseteq^{Pr} \mathcal{T}\|_\emptyset N'$. So neither $\sqsubseteq_B^{Pr}$ nor $\sqsubseteq^{Pr}$ are precongruences for $\|_\emptyset$.

A common solution to the problem of a preorder $\sqsubseteq$ not being a precongruence for certain operators is to instead consider its *congruence closure*, defined as the largest precongruence contained in $\sqsubseteq$.

In [30,15] the congruence closure of $\sqsubseteq^{Pr}$ is characterised as the so-called *NDFD* preorder $\sqsubseteq_{NDFD}$. Here $N \sqsubseteq_{NDFD} N'$ iff $N \sqsubseteq^{Pr} N'$ (characterised in the previous section) and moreover the divergence traces of N' are included in those of N. As remarked in [15], here it does not matter whether one requires congruence closure merely w.r.t. parallel composition and injective relabelling, or w.r.t. all operators of CSP (or CCSP, or anything in between).

Unlike $\sqsubseteq^{Pr}$, the preorder $\sqsubseteq^{J}$ is a precongruence for parallel composition. Although this has been proven already by Vogler [43], [22, in Appendix B] I provide a proof that bypasses the auxiliary notion of urgent transitions, and provides more details.

Proposition 2 ([43]) $\sqsubseteq^{J}$ is a precongruence for relabelling and abstraction.

Proof. This follows since $\mathscr{F}^{J}(f(N)) = \{(f(\sigma), X) \mid (\sigma, f^{-1}(X)) \in \mathscr{F}^{J}(N)\}$ and moreover $\mathscr{F}^{J}(\tau_I(N)) = \{(\tau_I(\sigma), X) \mid (\sigma, X \cup I) \in \mathscr{F}^{J}(N)\}$. Here $\tau_I(\sigma)$ is the result of pruning all I-actions from $\sigma \in \mathcal{A}^{\infty}$. □

Trivially, $\sqsubseteq^{J}$ also is a precongruence for $\sum a_i P_i$ and $a \triangleright \sum a_i P_i$.

The preorder $\sqsubseteq^{J}_{\mathcal{A}}$ can be seen to coincide with $\sqsubseteq^{Pr}_{\mathcal{A}}$, characterised as reverse inclusion of infinite and partial traces, and thus is a precongruence for the operators of CCSP. Leaving open the case $|\mathcal{A}\backslash B| = 1$, the preorders $\sqsubseteq^{J}_{B}$ with $|\mathcal{A}\backslash B| \geq 2$ fail to be precongruences for parallel composition.

Example 4 Take $b, c \notin B$. Let N, N' and $\mathcal{T}$ be as shown in Fig. 4. Then

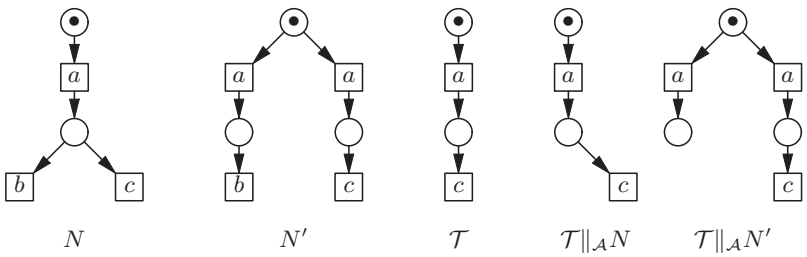


Fig. 4. The preorders $\sqsubseteq^{J}_{B}$ with $|\mathcal{A}\backslash B| \geq 2$ fail to be precongruences for parallel comp.

$N \equiv^{J}_{B} N'$, as $\mathscr{F}^{J}_{B}(N) = \mathscr{F}^{J}_{B}(N') = \{\varepsilon, ab, ac\}$. (Whether ε is included depends on whether $a \in B$.) Yet $\mathcal{T}\|_{\mathcal{A}} N \not\equiv^{J}_{B} \mathcal{T}\|_{\mathcal{A}} N'$, as $a \in \mathscr{F}^{J}_{B}(\mathcal{T}\|_{\mathcal{A}} N')$, yet $a \notin \mathscr{F}^{J}_{B}(\mathcal{T}\|_{\mathcal{A}} N)$.

Moreover, as illustrated below, the preorders $\sqsubseteq^{J}_{B}$ with $B \neq \emptyset$ and $|\mathcal{A}\backslash B| \geq 1$ fail to be precongruences for abstraction. In the next section I will show that, for $\mathcal{A}$ infinite and $B \neq \mathcal{A}$, the congruence closure of $\sqsubseteq^{J}_{B}$ for parallel composition, abstraction and relabelling is $\sqsubseteq^{J}$.

Example 5 Take $b \in B$ and $c \notin B$. Let N and N' be as shown in Fig. 5. Then $N \equiv^{J}_{B} N'$, as $\mathscr{F}^{J}_{B}(N) = \mathscr{F}^{J}_{B}(N') = \{\varepsilon, bc\}$. Yet $\tau_{\{b\}}(N) \not\equiv^{J}_{B} \tau_{\{b\}}(N')$, since $\varepsilon \in \mathscr{F}^{J}_{B}(\tau_{\{b\}}(N'))$, yet $\varepsilon \notin \mathscr{F}^{J}_{B}(\tau_{\{b\}}(N))$.

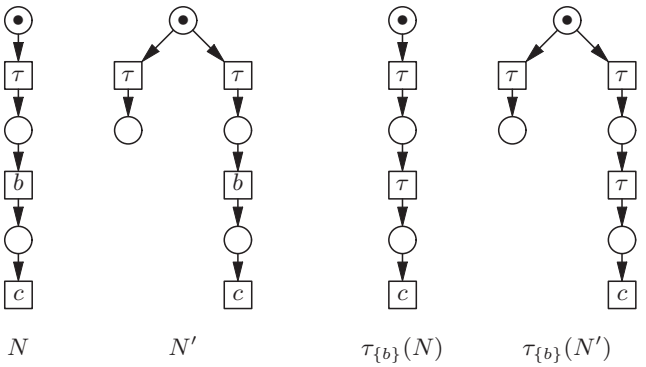


Fig. 5. The preorders $\sqsubseteq_B^J$ with $\emptyset \neq B \neq \mathcal{A}$ fail to be precongruences for abstraction

8 Must Testing

A *test* is a Petri net, but featuring a special action $w \notin \mathcal{A}_\tau$, not used elsewhere. This action is used to mark *success markings*: those in which w is enabled. If $\mathcal{T}$ is a test and N a net then $\tau_{\mathcal{A}}(\mathcal{T} \|_{\mathcal{A}} N)$ is also a test. An execution path of $\tau_{\mathcal{A}}(\mathcal{T} \|_{\mathcal{A}} N)$ is *successful* iff it contains a success marking.

Definition 9 A Petri net N *may pass* a test $\mathcal{T}$, notation N **may** $\mathcal{T}$, if $\tau_{\mathcal{A}}(\mathcal{T} \|_{\mathcal{A}} N)$ has a successful execution path. It *must pass* $\mathcal{T}$, notation N **must** $\mathcal{T}$, if each complete execution path of $\tau_{\mathcal{A}}(\mathcal{T} \|_{\mathcal{A}} N)$ is successful. It *should pass* $\mathcal{T}$, notation N **should** $\mathcal{T}$, if each finite execution path of $\tau_{\mathcal{A}}(\mathcal{T} \|_{\mathcal{A}} N)$ can be extended into a successful execution path.

Write $N \sqsubseteq_{\text{must}} N'$ if N **must** $\mathcal{T}$ implies N' **must** $\mathcal{T}$ for each test $\mathcal{T}$. The preorders $\sqsubseteq_{\text{may}}$ and $\sqsubseteq_{\text{should}}$ are defined similarly.

The may- and must-testing preorders stem from De Nicola & Hennessy [9], whereas should-testing was added independently in [5] and [36].

In the original work on testing [9] the CCS parallel composition $\mathcal{T}|N$ was used instead of the concealed CCSP parallel composition $\tau_{\mathcal{A}}(\mathcal{T} \|_{\mathcal{A}} N)$; moreover, only those execution paths consisting solely of internal actions mattered for the definitions of passing a test. The present approach is equivalent. First of all, restricting attention to execution paths of $\mathcal{T}|N$ consisting solely of internal actions is equivalent to putting $\mathcal{T}|N$ is the scope of a CCS restriction operator $\backslash\mathcal{A}$ [34], for that operator drops all transitions of its argument that are not labelled τ or w. Secondly, CCS features a complementary action $\bar{a}$ for each $a \in \mathcal{A}$, and one has $\bar{\bar{a}} = a$. For $\mathcal{T}$ a test, let $\overline{\mathcal{T}}$ denote the complementary test in which each action $a \in \mathcal{A}$ is replaced by $\bar{a}$; again $\overline{\overline{\mathcal{T}}} = \mathcal{T}$. It follows directly from the definitions of the operators involved that $\tau_{\mathcal{A}}(\mathcal{T} \|_{\mathcal{A}} N)$ is identical[3] to $(\overline{\mathcal{T}}|N)\backslash\mathcal{A}$. This proves the equivalence of the two approaches.

[3] The standard definition of | on Petri nets [28] is given only up to isomorphism. By choosing the names of places and transitions similar to those in the defintion of $\|_{\mathcal{A}}$ from [22, Appendix A] one can obtain $\tau_{\mathcal{A}}(\mathcal{T} \|_{\mathcal{A}} N) = (\overline{\mathcal{T}}|N)\backslash\mathcal{A}$.

Unlike may- and should-testing, the concept of must-testing is naturally parametrised with a completeness criterion, deciding what counts as a complete execution. To make this choice explicit I use the notation $\sqsubseteq^C_{\rm must}$, where C could be any of the completeness criteria surveyed in [25]. Since processes $\tau_{\mathcal{A}}(\mathcal{T}\|_{\mathcal{A}}N)$ (or $(\mathcal{T}|N)\backslash\mathcal{A}$) do not feature any actions other than τ and w, where w is used merely to point to the success states, the modifier $B \subseteq \mathcal{A}$ of a completeness criteria B-C has no effect, i.e., any two choices of this modifier are equivalent.

In the original work of [9] the default completeness criterion progress from Section 4 was employed. Interestingly, $\sqsubseteq^{Pr}_{\rm must}$ is a congruence for the operators of CCSP that does not preserve all linear time properties. It is strictly coarser than $\sqsubseteq_{NDFD}$. In fact, it is the coarsest precongruence for the CCSP parallel composition and injective relabelling that preserves those linear time properties that express that a system will eventually reach a state in which something [good] has happened [15]. (In [15], following [32], but deviating from the standard terminology of [1], such properties are called *liveness properties*.)

In this paper I investigate the must-testing preorder when taking justness as the underlying completeness criterion, $\sqsubseteq^J_{\rm must}$. Thm. 2 below shows that it can be characterised as the just failures preorder $\sqsubseteq^J$ of Section 6.

First note that Def. 9 can be simplified. When dealing with justness as completeness criterion, the word "complete" in Def. 9 is instantiated by "just" or "B-just", for some $B \subseteq \mathcal{A}$ (not including w). As the result is independent of B, one may take $B := \emptyset$. Since the labelling of a net has no bearing on its execution paths, or on whether such a path is $\emptyset$-just, or successful, one may now drop the operator τ_A from Def. 9 without affecting the resulting notion of must testing.

Theorem 2 $N \sqsubseteq^J_{\rm must} N'$ iff $N \sqsubseteq^J N'$.

Proof. The "if" direction is established in [22, Appendix C].

For "only if", suppose $N \sqsubseteq^J_{\rm must} N'$. Using Prop. 1, it suffices to show that $\mathscr{F}^J(N) \supseteq \mathscr{F}^J(N')$. Let $(\sigma, X) \in \mathscr{F}^J(N')$, where $\sigma = a_1a_2\ldots \in \mathcal{A}^\infty$ is a finite or infinite sequence of actions. Let $\mathcal{T}$ be the test displayed in Fig. 6. The drawing is for the case that $\sigma = a_1a_2\ldots a_n$ finite; in the infinite case, there is no need to display a_n separately. Now K **must** $\mathcal{T}$, for any net K, when using justness as completeness criterion, iff each $\emptyset$-just execution path of $\mathcal{T}\|_{\mathcal{A}}K$ is successful, which is the case iff $(\sigma, X) \notin \mathscr{F}^J(K)$. (In other words, $\mathcal{T}\|_{\mathcal{A}}K$ has an unsuccessful $\emptyset$-just execution path iff $(\sigma, X) \in \mathscr{F}^J(K)$. For the meaning of $(\sigma, X) \in \mathscr{F}^J(K)$ is that K has an execution path π with $\mathit{trace}(\pi) = \sigma$ such that $\ell_K(t) \in X \Rightarrow \neg\pi[t\rangle$.) Hence N' **must not** $\mathcal{T}$ and thus N **must not** $\mathcal{T}$, and thus $(\sigma, X) \in \mathscr{F}^J(N)$. □

Proposition 3 Let $\mathcal{A}$ be infinite and $B \neq \mathcal{A}$. Then $\sqsubseteq^J$ is the congruence closure of $\sqsubseteq^J_B$ for parallel composition, abstraction and injective relabelling.

Proof. Pick an action $w \in \mathcal{A}\backslash B$. Assume $N \not\sqsubseteq^J N'$. By applying an injective relabelling, one can assure that w does not occur in N or N'. Let $(\sigma, X) \in \mathscr{F}^J(N')$, yet $(\sigma, X) \notin \mathscr{F}^J(N)$, with $w \notin X$. Let T be the net of Fig. 6. Then, writing $A := \mathcal{A}\backslash\{w\}$, $(\sigma, \mathcal{A}) \in \mathscr{F}^J(\mathcal{T}\|_A N')$, yet $(\sigma, \mathcal{A}) \notin \mathscr{F}^J(\mathcal{T}\|_A N)$. Moreover, $(\rho, \mathcal{A}) \notin \mathscr{F}^J(\mathcal{T}\|_A N')$ and $(\rho, \mathcal{A}) \notin \mathscr{F}^J(\mathcal{T}\|_A N)$ for any $\rho \neq \sigma$ not containing the action

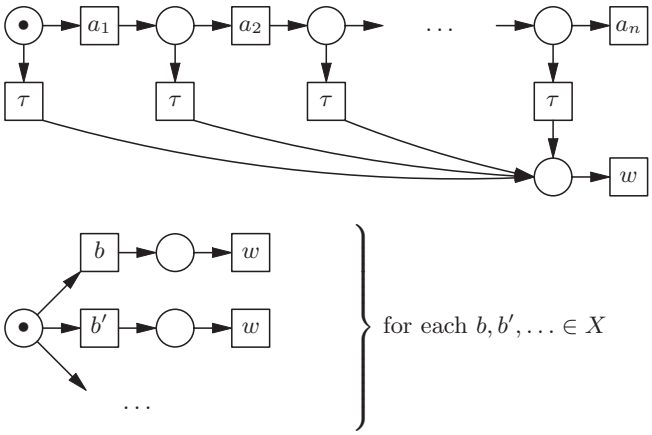


Fig. 6. Universal test for just must testing

w. Hence, applying the proof of Prop. 2, using that $A \cup \overline{B} = \mathcal{A}$, one has $(\varepsilon, \overline{B}) \in \mathscr{F}^J(\tau_A(\mathcal{T}\|_A N'))$, yet $(\varepsilon, \overline{B}) \notin \mathscr{F}^J(\tau_A(\mathcal{T}\|_A N))$. Thus $\varepsilon \in \mathscr{F}^J_B(\tau_A(\mathcal{T}\|_A N'))$, yet $\varepsilon \notin \mathscr{F}^J_B(\tau_A(\mathcal{T}\|_A N))$. It follows that $\tau_A(\mathcal{T}\|_A N) \not\sqsubseteq^J_B \tau_A(\mathcal{T}\|_A N')$. □

9 Timed must-testing

A timed form of must-testing was proposed by Vogler in [43]. Justness says that each transition that gets enabled must fire eventually, unless one of its necessary resources will be taken away. In Vogler's framework, each transition t must fire within 1 unit of time after it becomes enabled, even though it can fire faster. The implicit timer is reset each time t becomes disabled and enabled again, by another transition taken a token and returning it to one of the replaces of t. Since there is no lower bound on the time that may elapse before a transition fires, this view encompasses the same asynchronous behaviour of nets as under the assumption of justness.

Vogler's work only pertains to *safe* nets: those with the property that no reachable marking allocates multiple tokens to the same place. Here a marking is *reachable* if it occurs in some execution path. Transitions t with ${}^\bullet t = \emptyset$ are excluded. Although he only considered finite nets, here I apply his work unchanged to *finitely branching* nets: those in which only finitely many transitions are enabled in each reachable marking.

Definition 10 ([43]) A *continuous(ly timed) instantaneous description (CID)* of a net N is a pair (M, ξ) consisting of a marking M of N and a function ξ mapping the transitions enabled under M to $[0, 1]$; ξ describes the residual activation time of an enabled transition.

The initial CID is $\mathrm{CID}_0 = (M_0; \xi_0)$ with $\xi_0(t) = 1$ for all t with $M_0[t\rangle$.

One writes $(M, \xi)[\eta\rangle(M', \xi')$ if one of the following cases applies:

(1) $\eta = t \in T$, $M[t\rangle M'$, $\xi'(t) := \xi(t)$ for those transitions t enabled under $M - {}^\bullet t$ and $\xi'(t) := 1$ for the other transitions enabled under M'.
(2) $\eta = r \in \mathbb{R}^+$, $r \leq \min(\xi)$, $M' = M$ and $\xi' = \xi - r$.

A *timed execution path* π is an alternating sequence of CIDs and elements $t \in T$ or $r \in \mathbb{R}^+$, defined just like an execution path in Def. 6. Let $\zeta(\pi) \in \mathbb{R} \cup \{\infty\}$ be the sum of all time steps in a timed execution path π, the *duration* of π.

A *timed test* is a pair $(\mathcal{T}, D)$ of a test $\mathcal{T}$ and a *duration* $D \in \mathbb{R}_0^+$. A net *must pass* a timed test $(\mathcal{T}, D)$, notation N **must** $(\mathcal{T}, D)$, if each timed execution path π with $\zeta(\pi) > D$ contains a transition labelled w. Write $N \sqsubseteq_{\text{must}}^{\text{timed}} N'$ if N **must** $(\mathcal{T}, D)$ implies N' **must** $(\mathcal{T}, D)$ for each timed test $(\mathcal{T}, D)$.

Vogler shows that the preorder $\sqsubseteq_{\text{must}}^{\text{timed}}$ is strictly finer than $\sqsubseteq^J$. In fact, although $\tau.a.\mathbf{0} \equiv^J a.\mathbf{0}$, one has $\tau.a.\mathbf{0} \not\equiv_{\text{must}}^{\text{timed}} a.\mathbf{0}$, since only the latter process must pass the timed test $(a.w, 2)$. Here I use that each of the actions τ, a and w may take up to 1 unit of time to occur. A statement $N \sqsubseteq_{\text{must}}^{\text{timed}} N'$ says that N' is *faster* than N, in the sense that composed with a test it is guaranteed to reach success states in less time than N.

Here I show that when abstracting from the quantitative dimension of timed must-testing, it exactly characterises $\sqsubseteq^J$.

Definition 11 A net *must eventually pass* a test $\mathcal{T}$ if there exists a $D \in \mathbb{R}_0^+$ such that N **must** $(\mathcal{T}, D)$. Write $N \sqsubseteq_{\text{must}}^{\text{ev.}} N'$ if when N must eventually pass a test $\mathcal{T}$, then so does N'.

Theorem 3 Let N, N' be finitely branching safe nets. Then $N \sqsubseteq_{\text{must}}^{\text{ev.}} N'$ iff $N \sqsubseteq^J N'$.

A proof can be found in [22, Appendix D].

10 Conclusion

The just failures preorder $\sqsubseteq^J$ was introduced by Walter Vogler [43] in 2002. Since then it has not received much attention in the literature, and has not been used as the underlying semantic principle justifying actual verifications. In my view this can be seen as a fault of the subsequent literature, as $\sqsubseteq^J$ captures exactly what is needed—no more and no less—for the verification of safety and liveness properties of realistic systems.

I substantiate this claim by pointing out that $\sqsubseteq^J$ is the coarsest preorder preserving safety and liveness properties when assuming justness, that is a congruence for basic process algebra operators, such as the partially synchronous parallel composition, abstraction from internal actions, and renaming. As argued in [25,18,24,17], justness is better motivated and more suitable for applications than competing completeness criteria, such as progress or the many notions of fairness surveyed in [24].

Moreover, I adapt the well-known must-testing preorder of De Nicola & Hennessy [9], by using justness as the underlying completeness criterion, instead of

the traditional choice of progress. By showing that the resulting must-testing preorder $\sqsubseteq^{J}_{\text{must}}$ coincides with $\sqsubseteq^{J}$ I strengthen the case that this is a natural and fundamental preorder.

This conclusion is further strengthened by my result that it also coincides with a qualitative version $\sqsubseteq^{\text{ev.}}_{\text{must}}$ of the timed must-testing preorder $\sqsubseteq^{\text{timed}}_{\text{must}}$ of Vogler [43]. (Although $\sqsubseteq^{\text{timed}}_{\text{must}}$ and $\sqsubseteq^{J}$ stem from the same paper [43], this connection was not made there.)

All this was shown in the setting of Petri nets extended with read arcs, and therefore also applies to the settings of standard process algebras such as CCS, CSP or ACP. Since I cover read arcs, it also applies to process algebras enriched with signalling, an operator that extends the expressiveness of standard process algebras and is needed to accurately model mutual exclusion. I leave it for future work to explore these matters for probabilistic models of concurrency, or other useful extensions.

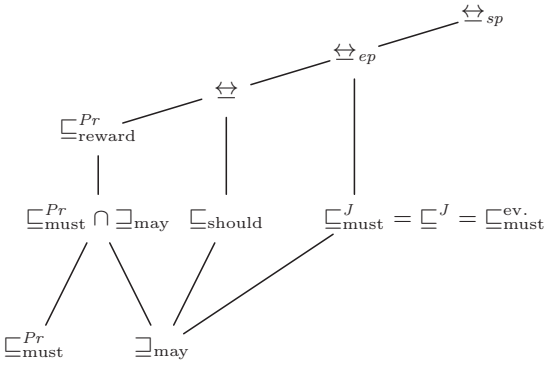
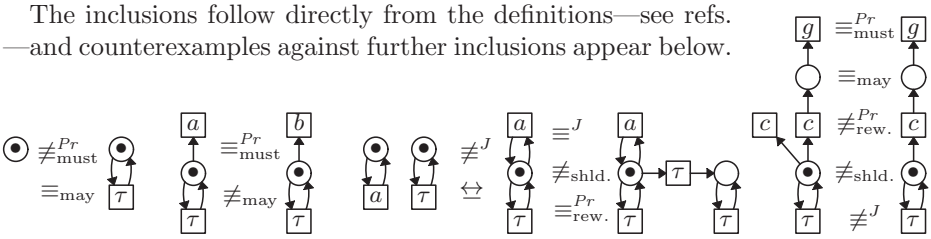


Fig. 7. A spectrum of testing preorders and bisimilarities preserving liveness properties

Fig. 7 situates $\sqsubseteq^{J}_{\text{must}}$ w.r.t. the some other semantic preorders from the literature. The lines indicate inclusions. Here $\sqsubseteq^{Pr}_{\text{must}}$, $\sqsubseteq_{\text{may}}$ and $\sqsubseteq_{\text{should}}$ are the classical must-, may- and should-testing preorders from [9] and [5,36]—see Def. 9—and $\sqsubseteq^{Pr}_{\text{reward}}$ is the reward-testing preorder introduced by me in [19]. The failures-divergences preorder of CSP [6,42], defined in a similar way as $\sqsubseteq^{J}_{\text{must}}$, coincides with $\sqsubseteq^{Pr}_{\text{must}}$ [9,19]. $\leftrightarrow$ denotes the classical notion of strong bisimilarity [34], and $\leftrightarrow_{ep}$, $\leftrightarrow_{sp}$ are essentially the only other preorders (in fact equivalences) that preserve linear time properties when assuming justness: the *enabling preserving bisimilarity* of [26] and the *structure preserving bisimilarity* of [16].

The inclusions follow directly from the definitions—see refs. —and counterexamples against further inclusions appear below.



References

1. Alpern, B., Schneider, F.B.: Defining liveness. Infromation Processing Letters **21**(4), 181–185 (1985). https://doi.org/10.1016/0020-0190(85)90056-0
2. Apt, K.R., Francez, N., Katz, S.: Appraising fairness in languages for distributed programming. Distributed Computing **2**(4), 226–241 (1988). https://doi.org/10.1007/BF01872848
3. Bergstra, J.A.: ACP with signals. In: Grabowski, J., Lescanne, P., Wechler, W. (eds.) Proc. International Workshop on Algebraic and Logic Programming. LNCS, vol. 343, pp. 11–20. Springer (1988). https://doi.org/10.1007/3-540-50667-5_53
4. Bergstra, J.A., Klop, J.W.: Algebra of communicating processes with abstraction. Theor. Comput. Sci. **37**(1), 77–121 (1985). https://doi.org/10.1016/0304-3975(85)90088-X
5. Brinksma, E., Rensink, A., Vogler, W.: Fair testing. In: Lee, I., Smolka, S.A. (eds.) Proc. 6th International Conference on Concurrency Theory, CONCUR'95. LNCS, vol. 962, pp. 313–327. Springer (1995). https://doi.org/10.1007/3-540-60218-6_23
6. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31**(3), 560–599 (1984). https://doi.org/10.1145/828.833
7. Busi, N., Pinna, G.M.: Non sequential semantics for contextual P/T nets. In: Billington, J., Reisig, W. (eds.) Proc. 17th Int. Conf. on Application and Theory of Petri Nets. LNCS, vol. 1091, pp. 113–132. Springer (1996). https://doi.org/10.1007/3-540-61363-3_7
8. Corradini, F., Di Berardini, M.R., Vogler, W.: Time and fairness in a process algebra with non-blocking reading. In: Nielsen, M., Kucera, A., Miltersen, P.B., Palamidessi, C., Tuma, P., Valencia, F.D. (eds.) Theory and Practice of Computer Science, SOFSEM'09. LNCS, vol. 5404, pp. 193–204. Springer (2009). https://doi.org/10.1007/978-3-540-95891-8_20
9. De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theor. Comput. Sci. **34**, 83–133 (1984). https://doi.org/10.1016/0304-3975(84)90113-0
10. Degano, P., De Nicola, R., Montanari, U.: CCS is an (augmented) contact free C/E system. In: Venturini Zilli, M. (ed.) Advanced School on Mathematical Models for the Semantics of Parallelism, 1986. LNCS, vol. 280, pp. 144–165. Springer (1987). https://doi.org/10.1007/3-540-18419-8_13
11. Dyseryn, V., van Glabbeek, R.J., Höfner, P.: Analysing mutual exclusion using process algebra with signals. In: Peters, K., Tini, S. (eds.) Proceedings Combined 24th International Workshop on Expressiveness in Concurrency and 14th Workshop on Structural Operational Semantics. EPTCS, vol. 255, pp. 18–34 (2017). https://doi.org/10.4204/EPTCS.255.2
12. van Glabbeek, R.J.: The linear time – branching time spectrum II; the semantics of sequential systems with silent moves. In: Best, E. (ed.) Proc. CONCUR'93, 4^{th} Int. Conf. on Concurrency Theory. LNCS, vol. 715, pp. 66–81. Springer (1993). https://doi.org/10.1007/3-540-57208-2_6
13. van Glabbeek, R.J.: A characterisation of weak bisimulation congruence. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity: Essays Dedicated to Jan Willem Klop on the Occasion of His 60th Birthday. LNCS, vol. 3838, pp. 26–39. Springer (2005). https://doi.org/10.1007/11601548_4

14. van Glabbeek, R.J.: The individual and collective token interpretations of Petri nets. In: Abadi, M., de Alfaro, L. (eds.) Proc. CONCUR'05, 16^{th} Int. Conf. on Concurrency Theory. LNCS, vol. 3653, pp. 323–337. Springer (2005). https://doi.org/10.1007/11539452_26
15. van Glabbeek, R.J.: The coarsest precongruences respecting safety and liveness properties. In: Calude, C., Sassone, V. (eds.) Proc. 6th IFIP TC 1/WG 2.2 Int. Conf. on Theoretical Computer Science, TCS'10; held as part of the *World Computer Congress*. IFIP, vol. 323, pp. 32–52. Springer (2010). https://doi.org/10.1007/978-3-642-15240-5_3, http://arxiv.org/abs/1007.5491
16. van Glabbeek, R.J.: Structure preserving bisimilarity, supporting an operational petri net semantics of CCSP. In: Meyer, R., Platzer, A., Wehrheim, H. (eds.) Proceedings Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday. LNCS, vol. 9360, pp. 99–130. Springer (2015). https://doi.org/10.1007/978-3-319-23506-6_9, http://arxiv.org/abs/1509.05842
17. van Glabbeek, R.J.: Ensuring liveness properties of distributed systems: Open problems. Journal of Logical and Algebraic Methods in Programming **109**, 100480 (2019). https://doi.org/10.1016/j.jlamp.2019.100480
18. van Glabbeek, R.J.: Justness: A completeness criterion for capturing liveness properties. In: Bojańczyk, M., Simpson, A. (eds.) Proc. 22st Int. Conf. on Foundations of Software Science and Computation Structures, FoSSaCS'19; held as part of ETAPS'19. LNCS, vol. 11425, pp. 505–522. Springer (2019). https://doi.org/10.1007/978-3-030-17127-8_29, https://arxiv.org/abs/1909.00286
19. van Glabbeek, R.J.: Reward testing equivalences for processes. In: Boreale, M., Corradini, F., Loreti, M., Pugliese, R. (eds.) Models, Languages, and Tools for Concurrent and Distributed Programming, Essays Dedicated to Rocco De Nicola on the occasion of his 65th Birthday, LNCS, vol. 11665, pp. 45–70. Springer (2019). https://doi.org/10.1007/978-3-030-21485-2_5, https://arxiv.org/abs/1907.13348
20. van Glabbeek, R.J.: Reactive temporal logic. In: Dardha, O., Rot, J. (eds.) Proc. Combined 27th Int. Workshop on Expressiveness in Concurrency and 17th Workshop on Structural Operational Semantics. EPTCS, vol. 322, pp. 51–68 (2020). https://doi.org/10.4204/EPTCS.322.6
21. van Glabbeek, R.J.: Modelling mutual exclusion in a process algebra with time-outs (2021), https://arxiv.org/abs/2106.12785
22. van Glabbeek, R.J.: Just Testing (2022), version of this paper extended with four appendices, https://arxiv.org/abs/2212.08829
23. van Glabbeek, R.J., Goltz, U., Schicke, J.W.: Abstract processes of place/transition systems. Information Processing Letters **111**(13), 626–633 (2011). https://doi.org/10.1016/j.ipl.2011.03.013, https://arxiv.org/abs/1103.5916
24. van Glabbeek, R.J., Höfner, P.: CCS: it's not fair! – fair schedulers cannot be implemented in CCS-like languages even under progress and certain fairness assumptions. Acta Informatica **52**(2-3), 175–205 (2015). https://doi.org/10.1007/s00236-015-0221-6, https://arxiv.org/abs/1505.05964
25. van Glabbeek, R.J., Höfner, P.: Progress, justness and fairness. ACM Computing Surveys **52**(4), 69 (August 2019). https://doi.org/10.1145/3329125, https://arxiv.org/abs/1810.07414
26. van Glabbeek, R.J., Höfner, P., Wang, W.: Enabling preserving bisimulation equivalence. In: Haddad, S., Varacca, D. (eds.) Proc. 32nd Int. Conference on Concurrency Theory, CONCUR'21. Leibniz International Proceedings

in Informatics (LIPIcs), vol. 203. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.CONCUR.2021.33, https://arxiv.org/abs/2108.00142
27. van Glabbeek, R.J., Vaandrager, F.W.: Petri net models for algebraic theories of concurrency. In: Bakker, J.W.d., Nijman, A.J., Treleaven, P.C. (eds.) Proc. PARLE, Parallel Architectures and Languages Europe, Vol. II. LNCS, vol. 259, pp. 224–242. Springer (1987). https://doi.org/10.1007/3-540-17945-3_13
28. Goltz, U.: CCS and Petri nets. In: Guessarian, I. (ed.) Proc. Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science. LNCS, vol. 469, pp. 334–357. Springer (1990). https://doi.org/10.1007/3-540-53479-2_14
29. Goltz, U., Mycroft, A.: On the relationship of CCS and Petri nets. In: Paredaens, J. (ed.) Proc. 11^{th} Colloquium on Automata, Languages and Programming, ICALP84. LNCS, vol. 172, pp. 196–208. Springer (1984). https://doi.org/10.1007/3-540-13345-3_18
30. Kaivola, R., Valmari, A.: The weakest compositional semantic equivalence preserving nexttime-less linear temporal logic. In: Cleaveland, R. (ed.) Proc. CONCUR'92. LNCS, vol. 630, pp. 207–221. Springer (1992). https://doi.org/10.1007/BFb0084793
31. Kindler, E., Walter, R.: Mutex needs fairness. Inf. Process. Lett. **62**(1), 31–39 (1997). https://doi.org/10.1016/S0020-0190(97)00033-1
32. Lamport, L.: Proving the correctness of multiprocess programs. IEEE Transactions on Software Engineering **3**(2), 125–143 (1977). https://doi.org/10.1109/TSE.1977.229904
33. Lamport, L.: Fairness and hyperfairness. Distributed Computing **13**(4), 239–245 (2000). https://doi.org/10.1007/PL00008921
34. Milner, R.: Communication and Concurrency. Prentice-Hall (1989), alternatively see *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980, https://doi.org/10.1007/3-540-10235-3
35. Montanari, U., Rossi, F.: Contextual nets. Acta Informatica **32**(6), 545–596 (1995). https://doi.org/10.1007/BF01178907
36. Natarajan, V., Cleaveland, R.: Divergence and fair testing. In: Fülöp, Z., Gécseg, F. (eds.) Proc. 22nd Int. Colloquium on Automata, Languages and Programming, ICALP'95. LNCS, vol. 944, pp. 648–659. Springer (1995). https://doi.org/10.1007/3-540-60084-1_112
37. Olderog, E.R.: Operational Petri net semantics for CCSP. In: Rozenberg, G. (ed.) Advances in Petri Nets 1987. LNCS, vol. 266, pp. 196–223. Springer (1987). https://doi.org/10.1007/3-540-18086-9_27
38. Olderog, E.R.: Nets, Terms and Formulas: Three Views of Concurrent Processes and Their Relationship. Cambridge Tracts in Theoretical Computer Science 23, Cambridge University Press (1991)
39. Olderog, E.R., Hoare, C.A.R.: Specification-oriented semantics for communicating processes. Acta Inf. **23**, 9–66 (1986). https://doi.org/10.1007/BF00268075
40. Reisig, W.: Understanding Petri Nets — Modeling Techniques, Analysis Methods, Case Studies. Springer (2013). https://doi.org/10.1007/978-3-642-33278-4
41. Roever, W.P.d., de Boer, F.S., Hannemann, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J.: Concurrency Verification: Introduction to Compositional and Noncompositional Methods, Cambridge Tracts in TCS, vol. 54. Cambridge University Press (2001)
42. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice-Hall (1997), http://www.comlab.ox.ac.uk/bill.roscoe/publications/68b.pdf

43. Vogler, W.: Efficiency of asynchronous systems, read arcs, and the MUTEX-problem. Theor. Comput. Sci. **275**(1-2), 589–631 (2002). https://doi.org/10.1016/S0304-3975(01)00300-0
44. Winskel, G.: A new definition of morphism on Petri nets. In: Fontet, M., Mehlhorn, K. (eds.) Proc. Symposium on Theoretical Aspects of Computer Science, STACS'84. LNCS, vol. 166, pp. 140–150. Springer (1984). https://doi.org/10.1007/3-540-12920-0_13

Model and Program Repair via Group Actions

Paul C. Attie[1] and William L. Cocke[1](✉)

School of Computer and Cyber Sciences, Augusta University, Augusta, GA, USA
pattie@augusta.edu, wcocke@augusta.edu

Abstract. Given a textual representation of a finite-state concurrent program P, one can construct the corresponding Kripke structure $\mathcal{M}$. However, the size of $\mathcal{M}$ can be exponentially larger than the textual size of P. This state explosion can make model checking properties of P via $\mathcal{M}$ expensive or even infeasible. The action of a symmetry group G on $\mathcal{M}$ can be used to produce a smaller Kripke structure $\overline{\mathcal{M}}$. Various authors have exploited the direct correspondence between $\mathcal{M}$ and $\overline{\mathcal{M}}$ to perform model checking. When the structure $\mathcal{M}$ does not satisfy a formula, one can look for a substructure that will satisfy the formula. We call this *substructure-repair*: identifying a substructure $\mathcal{N}$ of $\mathcal{M}$ that satisfies a given temporal logic formula.

In this paper we extend previous work by showing that repairs of $\overline{\mathcal{M}}$ lift to repairs of $\mathcal{M}$. In other words, we can repair a computer program P, which exhibits a high degree of symmetry, by repairing the smaller Kripke structure $\overline{\mathcal{M}}$ and then symmetrizing the corresponding program. To do this we arrange the substructures of $\mathcal{M}$ and $\overline{\mathcal{M}}$ into substructure lattices that are ordered by substructure inclusion. We show that the substructures of $\mathcal{M}$ preserved by G form a (sub)lattice that maps to the substructure lattice of $\overline{\mathcal{M}}$. When restricted to the lattice of substructures of $\mathcal{M}$ that are "maximal" with the action of G on $\mathcal{M}$, the above map is a lattice isomorphism.

These results enable us to repair $\overline{\mathcal{M}}$ and then to lift the repair to $\mathcal{M}$. In cases where a program has a high degree of symmetry, such as in many concurrent programs, we can repair the program by repairing the small Kripke structure $\overline{\mathcal{M}}$.

Keywords: Model checking · symmetry reduction · model repair

1 Introduction

To model check a program P, one first constructs a Kripke structure $\mathcal{M}$. In general, the Kripke structure $\mathcal{M}$ is generated by all potential executions of P. The model checking problem for a program P w.r.t. a temporal logic formula φ is to verify that the Kripke structure $\mathcal{M}$ generated by the execution of P satisfies φ [8]. A major obstacle to model checking a concurrent program via its Kripke structure is *state explosion*: in general, the size of $\mathcal{M}$ is exponential in the number of processes n. As studied by Emerson and Sistla [18] and extended by others [10,14,21], the use of *symmetry reduction* to ameliorate state-explosion

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 520–540, 2023.
https://doi.org/10.1007/978-3-031-30829-1_25

can yield a significant reduction in the complexity of model checking $\mathcal{M} \models \varphi$ when both $\mathcal{M}$ and φ have a high degree of symmetry in the process index set $\{1, \ldots, n\}$.

For a Kripke structure $\mathcal{M}$, we capture the symmetry of $\mathcal{M}$ using the group G of automorphisms of both $\mathcal{M}$ and φ. The quotient structure $\overline{\mathcal{M}} = \mathcal{M}/G$ of $\mathcal{M}$ by G often has significantly fewer states than $\mathcal{M}$. Since $\overline{\mathcal{M}}$ can be computed directly from the original P, we avoid the expensive computation of the large structure $\mathcal{M}$. Model checking $\overline{\mathcal{M}} \models \varphi$ is linear in the size of $\overline{\mathcal{M}}$ [8], so this provides significant savings if $\overline{\mathcal{M}}$ is small, i.e., if G is large.

If $\mathcal{M} \not\models f$, then we can search for a model $\mathcal{N}$ related to $\mathcal{M}$ such that $\mathcal{N} \models f$. In this paper we focus on *substructure-repair*: we require $\mathcal{N}$ to be a substructure of $\mathcal{M}$. The key idea behind substructure-repair is to remove execution paths which violate required properties, e.g., paths that lead to a violation of mutual exclusion. We give examples in Section 6 of different properties and substructure repairs with respect to these properties. Substructure-repairs can always repair $\mathcal{M}$ w.r.t. all universal properties (those expressible using universal path quantification [26]).[1]

1.1 Our Contributions

We present a theory of substructures of Kripke structures. Using this theory we establish an evaluation preserving correspondence between certain substructures of the original Kripke structure $\mathcal{M}$ and the substructures of the quotient structure $\overline{\mathcal{M}}$ (this is Theorem 2). This correspondence is a functorial form of bisimilarity between a certain lattice of substructures of $\mathcal{M}$ and the lattice of substructures of $\overline{\mathcal{M}}$. Hence for a given formula φ, substructure-repairs of $\overline{\mathcal{M}}$ with respect to φ can be lifted to substructure-repairs of $\mathcal{M}$ with respect to φ (this is Theorem 3). This correspondence of Kripke substructures lattices is of independent mathematical interest as an example of a monotone Galois connection.

We build on our theory to extend group theoretic model checking to *concurrent program repair*: given a concurrent program P that may not satisfy φ, modify P to produce a program that does satisfy φ. Given P, φ, and a group G that acts on both P and φ, our method directly computes the quotient $\mathcal{M}/G$ (following [18]), then repairs $\mathcal{M}/G$, using the algorithm of [2], and finally, extracts a correct program from the repaired structure.

The rest of the paper proceeds as follows: Section 3 contains the formal definition of Kripke structures and substructures. In Section 4, after briefly recalling group actions, we show how one can use a group to obtain a quotient $\overline{\mathcal{M}}$ of $\mathcal{M}$ and the repair correspondence between $\mathcal{M}$ and $\overline{\mathcal{M}}$. We extend our results to the repair of concurrent programs in Section 5. Section 6 presents some examples. In particular, we show that a structure $\mathcal{M}$ might have a nonempty repair even

[1] Existential path properties could be dealt with by first adding sufficient transitions to $\mathcal{M}$ so that the augmented structure now contains the desired paths. One can then perform substructure-repair so that universal path properties are also satisfied.

if the quotient $\overline{\mathcal{M}}$ does not. In Section 7 we examine what classes of Kripke structures and what types of formulae guarantee the existence of quotient based repairs.

2 Related Work

Our work combines model/program repair [5,25,29,32] and symmetry reductions via group actions [7, 10, 16, 18–22]. Le Goues et al. [25] provides a modern introduction to program repair; although their results generally relate to program repair based on the textual representation of the program. Our approach repairs a Kripke structure w.r.t. a computation tree logic (CTL) formula and uses that to repair the corresponding program.

2.1 Computation Tree Logic Repair

Buccafuri et. al. [5] posed the repair problem for CTL and solved it using abductive reasoning to generate repair suggestions that are verified by model checking. Jobstmann et. al. [29] and Staber et. al. [32] used game-based repair methods for programs and circuits, although their method is complete for invariants only.

Chatzieleftheriou et. al. [6] repair abstract structures, using Kripke modal transition systems and 3-valued CTL semantics. Von Essen and Jobstmann [23] present a game-based repair method which attempts to keep the repaired program close to the original faulty program, by also specifying a set of traces that the repair must leave intact.

The work of Attie et al. [2] establishes that repair by abstraction can avoid state explosion. However, repairs of abstracted structures do not always lift to repairs of the original structure. Within networks, Namjoshi and Trefler [30] have shown that a combination of abstraction and group actions can be used to produce smaller structures.

2.2 Group theoretic model checking

Group theoretic approaches to symmetry-reduction in model checking began in 1995 with work by Emerson and a collection of coauthors [7, 10, 14, 16, 18–22] compute the quotient $\mathcal{M}/G$ and model check $\mathcal{M}/G$, instead of the original (much larger) structure $\mathcal{M}$. The group theoretic approach to model checking works because $\mathcal{M}$ and $\mathcal{M}/G$ are bisimilar with respect to certain formulae.

A requirement for group theoretic model checking or repair is calculating the group of symmetries in question. We will see that larger groups of symmetries result in smaller quotient models. Clarke et al. [7] showed that calculating the orbit of a group action, a part of model checking via symmetry, is at least as difficult as graph isomorphism. However, in many practical cases concurrent programs have a natural symmetry by swapping certain processes. Hence many concurrent programs have a small known symmetry group in advance. Donaldson

and Miller [11] showed that there is a process to build a larger symmetry group for a program from a smaller symmetry group.

A related approach is the use of structural methods to express symmetric designs, e.g., parameterized systems, where processes are all instances of a common template (possibly with a distinguished controller process) [1, 9, 24], and rings of processes, where all communication is between a process and its neighbors in the ring [9, 15, 17].

3 Temporal Logic and Kripke Structures

Computation tree logic (CTL) is a propositional branching-time temporal logic used to model the possible computational branches taken by a system [12, 13]. The semantics of CTL are defined with respect to a Kripke structure.

Definition 1 (Kripke structure). *A* ***Kripke structure*** $\mathcal{M}$ *is a tuple* (S, S_0, T, L, AP) *where* S *is a finite set of states,* $S_0 \subseteq S$ *is a set of initial states,* $T \subseteq (S \times S)$ *is a transition relation,* AP *is a finite set of atomic propositions, and* $L : S \to 2^{AP}$ *is a labeling function that associates each state* $s \in S$ *with a subset of atomic propositions, namely those that hold in state* s*.*

We require that $\mathcal{M}$ be total: $\forall s \in S, \exists t \in S : (s,t) \in T$, and that $S \neq \emptyset$ implies $S_0 \neq \emptyset$. Also, different states have different labels: $s \neq t \Rightarrow L(s) \neq L(t)$. We admit the empty Kripke structure, i.e., $S = \emptyset$, due to mathematical necessity.

When referring to the constituents of $\mathcal{M} = (S, S_0, T, L, AP)$, we write $\mathcal{M}_S$, $\mathcal{M}_{S_0}$, $\mathcal{M}_T$, $\mathcal{M}_L$, and $\mathcal{M}_{AP}$ respectively. State t is a *successor* of state s in M iff $(s,t) \in T$. We will write $s \to t$ in this case. A path π in $\mathcal{M}$ is a (finite or infinite) sequence of states, $\pi = s_0, s_1, \ldots$, such that $\forall i \geq 0 : (s_i, s_{i+1}) \in T$.

To model the behavior of a concurrent program $P = P_1, \ldots, P_n$, we define a special type of Kripke structure: a *multiprocess Kripke structure* is one in which the set of atomic propositions AP is partitioned into disjoint subsets $AP_1, \ldots, AP_n$, states have the form $(s_1, \ldots, s_n)$ and transitions T are partitioned into disjoint subsets $T_1, \ldots, T_n$. The set of atomic propositions "owned" by P_i is denoted by AP_i: they can only be changed by P_i, but can be read by other processes. The local state of P_i is written as s_i, and is labelled by the subset of AP_i whose propositions are true in s_i. Then, the truth value of $p \in AP_i$ in global state $(s_1, \ldots, s_n)$ is given by its value in local state s_i. T_i gives the transitions of process P_i, which are denoted as $s \xrightarrow{i} t$. For state $s = (s_1, \ldots, s_n)$, define $s\restriction i = s_i$, and $s\downarrow i = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$. We then require $s\downarrow i = t\downarrow i$ for every transition $s \xrightarrow{i} t$, i.e., transitions by P_i do not change atomic propositions of other processes.

A CTL formula φ is evaluated (i.e., is true or false) in a state s of a Kripke structure $\mathcal{M}$ [13]. We write $\mathcal{M}, s \models \varphi$ when s is true in state s of structure $\mathcal{M}$, and write $\mathcal{M} \models \varphi$ to abbreviate $\forall s_0 \in S_0 : \mathcal{M}, s_0 \models \varphi$, i.e., φ holds in all initial states of $\mathcal{M}$. The formal definition of $\models$, proceeds by induction on the structure of CTL formulae [12, 13] and is omitted for space reasons.

Example 1 (Example Box) The "Box" Kripke structure in Figure 1 has 4 states and transitions as shown. Its set of atomic propositions is empty, and so all states have empty labels, as indicated by "()". There is a natural group acting on this Kripke structure, i.e., the group generated by the action which exchanges the state **s1** with **s2**, and the state **t1** with **t2**.

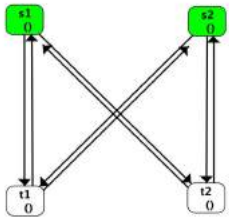


Fig. 1. The Box Kripke structure.

The theory of substructures presented below is motivated by the concept of a substructure-repair of a structure $\mathcal{M}$ with respect to a formula f, i.e., a substructure $\mathcal{N}$ of $\mathcal{M}$ such that $\mathcal{N} \models f$.

Definition 2 (Substructure, $\leq$). *Given Kripke structures $\mathcal{M}$ and $\mathcal{N}$, we say that $\mathcal{N}$ is a **substructure** of $\mathcal{M}$, denoted $\mathcal{N} \leq \mathcal{M}$, iff the following all hold:*

1. $\mathcal{N}_S \subseteq \mathcal{M}_S$.
2. $\mathcal{N}_{S_0} = \mathcal{M}_{S_0} \cap \mathcal{M}'_S$.
3. $\mathcal{N}_T \subseteq \mathcal{M}_T$.
4. $\mathcal{N}_{AP} = \mathcal{M}_{AP}$.
5. $\mathcal{N}_L = \mathcal{M}_L \restriction S'$ *(where $\restriction$ denotes domain restriction).*
6. *For all $s \in \mathcal{N}_S$ there is a $t \in \mathcal{N}_S$ such that $(s,t) \in \mathcal{N}_T$, i.e., $\mathcal{N}$ is total.*

For mathematical necessity in what follows, we allow for the 'empty' substructure. We do not, however, accept an empty substructure as a valid repair. It is immediate that $\leq$ is a reflexive partial order. Lemmas 1 and 2 below imply that the substructures of $\mathcal{M}$ can be regarded as a lattice, with join and meet operations as follows.

Lemma 1. *Let $\mathcal{M}$ be a Kripke structure and suppose that $\mathcal{N}$ and $\mathcal{N}'$ are substructures of $\mathcal{M}$. Then*

$$\mathcal{N} \vee \mathcal{N}' = (\mathcal{N}_S \cup \mathcal{N}'_S, \mathcal{N}_{S_0} \cup \mathcal{N}'_{S_0}, \mathcal{N}_T \cup \mathcal{N}'_T, \mathcal{M}_L \restriction (\mathcal{N}_S \cup \mathcal{N}'_S), \mathcal{M}_{AP})$$

is the smallest substructure of $\mathcal{M}$ containing both $\mathcal{N}$ and $\mathcal{N}'$.

Given a nonempty finite set $X = \{X_0, X_1, \ldots, X_n\}$ of substructures of $\mathcal{M}$, we define the structure $\bigvee X = X_0 \vee X_1 \vee \cdots \vee X_n$.

Lemma 2. *Let $\mathcal{M}$ be a Kripke structure and suppose that $\mathcal{N}$ and $\mathcal{N}'$ are substructures of $\mathcal{M}$. Then there exists a largest substructure of $\mathcal{M}$ contained in both $\mathcal{N}$ and $\mathcal{N}'$.*

Definition 3 (Join, Meet of Substructures). *Let $\mathcal{N}$ and $\mathcal{N}'$ be two substructures of $\mathcal{M}$. The **join** of $\mathcal{N}$ and $\mathcal{N}'$, written $\mathcal{N} \vee \mathcal{N}'$, is the smallest substructure of $\mathcal{M}$ containing both $\mathcal{N}$ and $\mathcal{N}'$. The **meet** of $\mathcal{N}$ and $\mathcal{N}'$, written $\mathcal{N} \wedge \mathcal{N}'$, is the largest substructure of $\mathcal{M}$ contained in both $\mathcal{N}$ and $\mathcal{N}'$.*

The join $\mathcal{N} \vee \mathcal{N}'$ has a simple description as given in Lemma 1. However, the meet $\mathcal{N} \wedge \mathcal{N}'$, while well-defined, does not have such a simple description. It is possible that for two substructures $\mathcal{N}$ and $\mathcal{N}'$ of a Kripke structure $\mathcal{M}$, there are no non-empty substructures contained in both $\mathcal{N}$ and $\mathcal{N}'$. Hence the largest substructure contained in both $\mathcal{N}$ and $\mathcal{N}'$ could be empty.

We can now define a lattice of substructures $\Lambda_{\mathcal{M}}$ for a given structure $\mathcal{M}$.

Definition 4 (Lattice of Substructures). *Given a Kripke structure $\mathcal{M}$ the **lattice of substructures of** $\mathcal{M}$ is $\Lambda_{\mathcal{M}} = (\{\mathcal{N} : \mathcal{N}$ is a substructure of $\mathcal{M}\}, \leq)$ where the meet and join in $\Lambda_{\mathcal{M}}$ are as given in Definition 3.*

4 Quotient Structures

We capture the symmetry in a Kripke structure $\mathcal{M}$ with the notion of *state-mapping*: a graph isomorphism on $\mathcal{M}$ which preserves initial states. State-mappings also preserve paths since they are isomorphisms. We ignore for now the labelling function $\mathcal{M}_L$, i.e., which atomic propositions hold in which states, and concern ourselves only with the graph structure of $\mathcal{M}$. Since the atomic proposition labelling obviously affects the truth of CTL formulae in states of $\mathcal{M}$, it must be accounted for. We do this below using the notion of G-invariant CTL formula. Thus, we decompose the symmetry characerization of $\mathcal{M}$ into two separate concerns: the graph structure of $\mathcal{M}$, handled using state-mapping, and the atomic proposition labelling of states of $\mathcal{M}$, handled using G-invariant CTL formulae.

A type of symmetry of particular interest is the symmetry of a multiprocess Kripke structure w.r.t. the process indices $1, \ldots, n$ of the corresponding concurrent program $P_1 \parallel \cdots \parallel P_n$, as we illustrate below. Our theory, however, applies to Kripke structures in general.

4.1 Groups Acting on Kripke Structures

Definition 5. *A state-mapping of $\mathcal{M}$ is a graph isomorphism of the state-space of $\mathcal{M}$ such that its restriction to the initial states is also an isomorphism, i.e., takes initial states to initial states. Formally, for a Kripke structure $\mathcal{M}$, a **state-mapping** of $\mathcal{M}$ is a bijection $f : \mathcal{M}_S \to \mathcal{M}_S$ such that:*

- $f(\mathcal{M}_{S_0}) = \mathcal{M}_{S_0}$;
- *For states $s, t \in \mathcal{M}_S$ we have that $(s,t) \in \mathcal{M}_T \iff (f(s), f(t)) \in \mathcal{M}_T$.*

The set of all state-mappings of $\mathcal{M}$ forms a group. This means that the composition of any two state-mappings is another state-mapping and for any state-mapping f on $\mathcal{M}$ there is another state-mapping g on $\mathcal{M}$ such that $f(g(s)) = s$ and $g(f(s)) = s$. We refer to the manuscripts by Issacs [27,28], and Serre [31] for a more in-depth introduction to group theory.

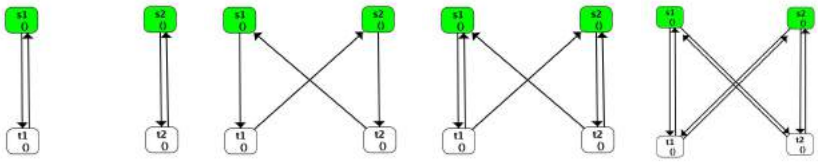
Definition 6 (G-closed). *For a group G of state-mappings of a Kripke structure $\mathcal{M}$, a substructure $\mathcal{N}$ of $\mathcal{M}$ is called **G-closed** if G is a group of state-mappings of $\mathcal{N}$, i.e., for every $g \in G$ and $s \in \mathcal{N}_S$ we have $g(s) \in \mathcal{N}_S$.*

Lemma 3. *Let $\mathcal{M}$ be a Kripke structure and let G be a group of state mappings of $\mathcal{M}$. Let $\mathcal{N}$, $\mathcal{N}'$ be two G-closed substructures of $\mathcal{M}$. Then $\mathcal{N} \vee \mathcal{N}'$ and $\mathcal{N} \wedge \mathcal{N}'$ are both G-closed.*

By Lemma 3, we see that the G-closed substructures of $\mathcal{M}$ form a sublattice of $\Lambda_{\mathcal{M}}$. This is a proper sublattice in that the meet and join operations are the same as those of $\Lambda_{\mathcal{M}}$.

Definition 7 (Lattice of G-closed substructures). *Given a Kripke structure $\mathcal{M}$ and a group G of state mappings of $\mathcal{M}$, the poset of G-closed substructures of $\mathcal{M}$ forms a lattice. We call this the* ***lattice of G-closed substructures of $\mathcal{M}$*** *and write it as $\Lambda_{\mathcal{M},G}$.*

Example 1 (Example Box). Let $\mathcal{M}$ be Example Box, i.e., the Kripke structure presented in Figure 1. Let g be the map that simultaneously switches $\mathbf{s_1}$ and $\mathbf{s_2}$, and switches $\mathbf{t_1}$ and $\mathbf{t_2}$, i.e., $g(\mathbf{s_1}) = \mathbf{s_2}$, $g(\mathbf{s_2}) = \mathbf{s_1}$, $g(\mathbf{t_1}) = \mathbf{t_2}$, $g(\mathbf{t_2}) = \mathbf{t_1}$. Let G be the group consisting of g and the identity map on $\mathcal{M}_S$. We note that G is not the entire group of state-mappings of $\mathcal{M}$. The structure $\mathcal{M}$ has 10 G-closed substructures, including the empty structure. We present some of these structures in Figure 2.



(a) (b) (c) (d)

Fig. 2. Four G-closed substructures of Example Box. Where G is the group generated by the simultaneous swapping of indexes of both the $\mathbf{s_i}$ and the $\mathbf{t_i}$. Note that each of the structures is a substructure of the substructure to the right. Looking ahead to Definition 10, only the entire structure (d) is G-maximal.

4.2 Constructing the Quotient structure

Given a group G of state-mappings of a structure $\mathcal{M}$, we want to construct a quotient structure $\mathcal{M}/G$. However, as noted, state-mappings do not contain any information about $\mathcal{M}_L$. To remedy this situation, we need a function that assigns a representative to each orbit of G, where for $s \in \mathcal{M}_S$ the orbit of s is $\{g(s) : g \in G\}$.

Definition 8 (Representative map). *Let $\mathcal{M}$ be a Kripke structure and suppose that G is a group of state-mappings of $\mathcal{M}$. A* ***representative map*** *of $\mathcal{M}$ with respect to G is a function $\vartheta_G : \mathcal{M}_S \to \mathcal{M}_S$ satisfying the following:*

- *For all $s, s' \in \mathcal{M}_S$, if there is some $g \in G$ such that $g(s) = s'$ then $\vartheta_G(s) = \vartheta_G(s')$. (respects orbits)*
- *For all $s, s' \in \mathcal{M}_S$, if there is no $g \in G$ such that $g(s) = s'$ then $\vartheta_G(s) \neq \vartheta_G(s')$. (separates orbits)*
- *For all $s \in \mathcal{M}_S$, we have that $\vartheta_G(\vartheta_G(s)) = \vartheta_G(s)$, i.e., each orbit has a stable representative. (idempotent)*

We define $\vartheta_G(S) = \{\vartheta_G(s) \mid s \in S\}$.

Definition 9 (Quotient structure). *Given a Kripke structure $\mathcal{M}$, a group G of state-mappings of $\mathcal{M}$, and a representative map ϑ_G of $\mathcal{M}$ with respect to G, we define the **quotient structure** $\overline{\mathcal{M}} = \mathcal{M}/(G, \vartheta_G)$ of $\mathcal{M}$ with respect to G and ϑ_G as follows, where we write $\overline{s}$, $\overline{t}$ for $\vartheta_G(s)$, $\vartheta_G(t)$, respectively:*

- *$\overline{\mathcal{M}}_S = \vartheta_G(\mathcal{M}_S)$, i.e., the states of $\overline{\mathcal{M}}$ are the image under ϑ_G of the states of $\mathcal{M}$.*
- *$\overline{\mathcal{M}}_T$ consists of all $(\overline{s}, \overline{t})$ such that there exist $s \in \mathcal{M}_S$ with $\vartheta_G(s) = \overline{s}$ and $t \in \mathcal{M}_S$ with $\vartheta_G(t) = \overline{t}$ such that $(s, t) \in \mathcal{M}_T$.*
- *$\overline{\mathcal{M}}_{S_0} = \vartheta_G(\mathcal{M}_{S_0})$, i.e., the initial states of $\overline{\mathcal{M}}$ are the image under ϑ_G of the initial states of $\mathcal{M}$.*
- *$\overline{\mathcal{M}}_L(\overline{s}) = \mathcal{M}_L(\overline{s})$, i.e., the label of a state in $\overline{\mathcal{M}}$ is the same as its label in $\mathcal{M}$.*
- *$\overline{\mathcal{M}}_{AP} = \mathcal{M}_{AP}$, i.e., $\overline{\mathcal{M}}$ has the same atomic propositions as $\mathcal{M}$.*

Thus *the states of a quotient structure correspond exactly to the orbits of states of the original structure* under the group of state mappings. For transitions, we have a slightly more subtle correspondence. Consider the following examples:

Example 2. In Figure 3 we demonstrate the correspondence between Kripke structures, G-closed substructures, and their quotients. In the figure, we present a multiprocess Kripke structure $\mathcal{M}$ corresponding to two concurrent processes P_1 (atomic propositions and transitions in blue) and P_2 (atomic propositions and transitions in red). The group G of state mappings swaps the indexes of the processes. This structure has a G-closed substructure $\mathcal{N}$ constructed by removing the 'center' state $\mathbf{u_0}$. Define ϑ_G to take the 'left-most' state in the orbit, i.e., $\vartheta_G(\mathbf{t_1}) = \mathbf{t_0}$, $\vartheta_G(\mathbf{t_5}) = \mathbf{t_2}$, $\vartheta_G(\mathbf{u_0}) = \mathbf{u_0}$, $\vartheta_G(\mathbf{t_6}) = \mathbf{t_3}$, $\vartheta_G(\mathbf{t_4}) = \mathbf{t_4}$. The quotient structure $\mathcal{M}/(G, \vartheta_G)$ appears in the top right. While the quotient structure is isomorphic to a substructure of $\mathcal{M}$, this is not always the case. (See Figure 6 in Example 5 for an example where the quotient gains a new transition.) The quotient structure $\mathcal{N}/(G, \vartheta_G|\mathcal{N}_S)$ appears in the bottom right.

Example 3 (Example Box). Let $\mathcal{M}$ and G be as in Example 1. Let ϑ_G be defined by $\vartheta_G(\mathbf{s_1}) = \vartheta_G(\mathbf{s_2}) = \mathbf{s_1}$ and $\vartheta_G(\mathbf{t_1}) = \vartheta_G(\mathbf{t_2}) = \mathbf{t_1}$. Then the quotient structure $\mathcal{M}/(G, \vartheta_G)$ has exactly 2 states, $\mathbf{s_1}$ and $\mathbf{t_1}$ with transitions $(\mathbf{s_1}, \mathbf{t_1}), (\mathbf{t_1}, \mathbf{s_1})$. Also, the G-closed substructure substructures of $\mathcal{M}$ given in Figure 2 (a), (b), and (c) also map to this quotient structure via $\mathcal{N} \to \mathcal{N}/(G, \vartheta_G)$. Note that the transition $(\mathbf{t_1}, \mathbf{s_1})$ is present in the quotient, but is not present, for example, in the structure of Figure 2 (b). However, the "corresponding" transition $(\mathbf{t_2}, \mathbf{s_1})$ is present in Figure 2 (b).

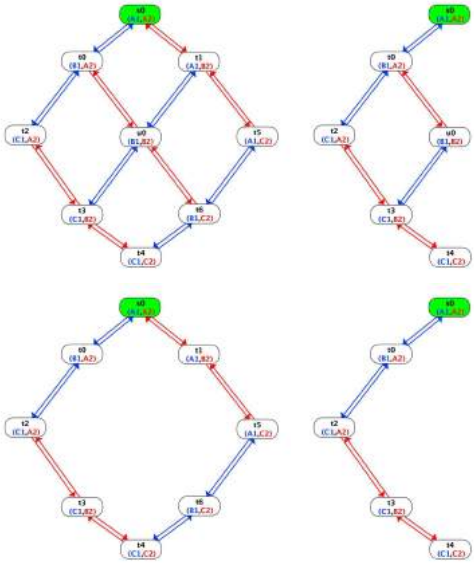


Fig. 3. As discussed in Example 2, we have a Kripke structure in the top left and a G-closed substructure in the bottom left. On the right, we have the quotients of the whole structure (top) and the G-closed substructure (bottom).

In the sequel, we fix a Kripke structure $\mathcal{M}$, a group G of state mappings of $\mathcal{M}$, and a representative map ϑ_G of $\mathcal{M}$ with respect to G.

Example 3 shows that **many G-closed substructures can have the same quotient structure**, and also that, in general, a transition in the quotient may not itself be present in the original structure. We show, however, in Theorem 1 below that a "corresponding" transition is guaranteed to be present in the original structure. These corresponding transitions can be joined into a path which corresponds state-by-state to the path in the quotient. This "path correspondence" is what allows for model checking of $\mathcal{M}$ via model checking of $\overline{\mathcal{M}}$ and is formalized in the following theorem from Emerson and Sistla [18, **3.1**].

Theorem 1 (Path Correspondence Theorem). *There is a bidirectional correspondence between paths of $\mathcal{M}$ and paths of $\overline{\mathcal{M}}$. Formally we have the following:*

1. *If $x = s_0, s_1, s_2, \ldots$ is a path in $\mathcal{M}$, then $\overline{x} = \overline{s_0}, \overline{s_1}, \overline{s_2}, \ldots$ is a path in $\overline{\mathcal{M}}$ where $\overline{s_i} = \vartheta_G(s_i)$.*
2. *If $\overline{x} = \overline{s_0}, \overline{s_1}, \overline{s_2}, \ldots$ is a path in $\overline{\mathcal{M}}$, then for every state $s'_0 \in \mathcal{M}_S$ such that $\vartheta_G(s'_0) = \overline{s_0}$ there is a path $s'_0, s'_1, s'_2, \ldots$ in $\mathcal{M}$ such that $\vartheta_G(s'_i) = \overline{s_i}$.*

We now extend the path correspondence between $\mathcal{M}$ and $\overline{\mathcal{M}}$ to a correspondence between G-closed substructures of $\mathcal{M}$ and substructures of $\overline{\mathcal{M}}$. Define $\Psi : \Lambda_{\mathcal{M},G} \to \Lambda_{\overline{\mathcal{M}}}$, by $\Psi(\mathcal{N}) = \mathcal{N}/(G, \vartheta_G)$, so that Ψ maps a G-closed substructure $\mathcal{N}$ of $\mathcal{M}$ to a corresponding substructure of $\overline{\mathcal{M}}$. We call Ψ the *quotient map*.

Ψ establishes a join-semilattice homomorphism between $\Lambda_{\mathcal{M},G}$ and $\Lambda_{\overline{\mathcal{M}}}$ as we now show in the following series of lemmas.

Lemma 4. *For every substructure $\overline{\mathcal{N}}$ of $\overline{\mathcal{M}}$, there is a G-closed substructure $\mathcal{N}$ of $\mathcal{M}$ such that $\mathcal{N}/(G,\vartheta_G) = \overline{\mathcal{N}}$.*

Lemma 4 establishes that Ψ is surjective. We note that every substructure $\overline{\mathcal{N}}$ of $\overline{\mathcal{M}}$ defines a set of states of $\overline{\mathcal{M}}$, i.e., the orbits of the states in $\overline{\mathcal{N}}$. However, in general, the transitions of $\overline{\mathcal{N}}$ do not uniquely define transitions in $\overline{\mathcal{M}}$.

The next lemma demonstrates that Ψ is a homomorphism of the join-semilattices $\Lambda_{M,G}$ and $\Lambda_{\overline{\mathcal{M}}}$. We note that it is not a homomorphism of the lattices themselves because the meet of two G-closed structures mapping might be empty.

Lemma 5 (Quotient map respects join). *Let $\mathcal{N},\mathcal{N}' \in \Lambda_{\mathcal{M},G}$. Then*

$$\Psi(\mathcal{N} \vee \mathcal{N}') = \Psi(\mathcal{N}) \vee \Psi(\mathcal{N}').$$

As seen in Example 3, it is possible for multiple G-closed substructures of $\mathcal{M}$ to map to the same substructure of the quotient structure $\overline{\mathcal{M}}$. To obtain a single well-defined preimage for each substructure of the quotient structure, we introduce the concept of G-maximal. Recall that the join of G-closed substructures of $\mathcal{M}$ is G-closed.

Definition 10 (G-maximal). *A G-closed substructure $\mathcal{N}$ of $\mathcal{M}$ is G-maximal if*

$$\mathcal{N} = \bigvee_{\substack{\mathcal{N}' \in \Lambda_{\mathcal{M},G} \\ \mathcal{N}'/(G,\vartheta_G) \leq \mathcal{N}/(G,\vartheta_G)}} \mathcal{N}'.$$

That is, $\mathcal{N}$ is the join of all G-closed substructures of $\mathcal{M}$ whose quotient is a substructure of the quotient of $\mathcal{N}$ itself, namely of $\mathcal{N}/(G,\vartheta_G)$. A G-closed substructure $\mathcal{N}$ fails to be G-maximal exactly when there are states $s,t \in \mathcal{N}$, such that $(s,t) \in \mathcal{N}/(G,\vartheta_G)$, but (s,t) is not in $\mathcal{N}$.

Among all of the G-closed substructures in Figure 2 only the entire structure itself is G-maximal

Lemma 6. *Let $\mathcal{M}',\mathcal{M}''$ be two G-maximal substructures of $\mathcal{M}$. Then $\mathcal{M}' \vee \mathcal{M}''$ is G-maximal and $\mathcal{M}' \wedge \mathcal{M}''$ is G-maximal.*

Lemma 6 allows us to make the following definition.

Definition 11 (G-maximal lattice of substructures). *The set of G-maximal substructures of $\mathcal{M}$ forms a sublattice $\Lambda_{\mathcal{M},G-max}$ of $\Lambda_{\mathcal{M}}$.*

While in general the quotient map from $\Lambda_{\mathcal{M},G}$ to $\Lambda_{\overline{\mathcal{M}}}$ is always surjective, when restricted to $\Lambda_{\mathcal{M},G-\max}$, the map is injective and is a lattice isomorphism.

Theorem 2 (G-Maximal Lattice Correspondence). *The restriction of the quotient map Ψ to $\Lambda_{\mathcal{M},G-max}$ is an isomorphism from $\Lambda_{\mathcal{M},G-max}$ to $\Lambda_{\overline{\mathcal{M}}}$, i.e., between the lattice of G-maximal substructures of $\mathcal{M}$ and the lattice of structures of $\overline{\mathcal{M}}$.*

At this point, we would like to remind the reader of the various lattices that we have defined and how they relate to each other:

$$\underbrace{G\text{-maximal substructures}}_{\Lambda_{\mathcal{M},G-\max}} \subseteq \underbrace{G\text{-closed substructures}}_{\Lambda_{\mathcal{M},G}} \subseteq \underbrace{\text{All substructures}}_{\Lambda_{\mathcal{M}}}.$$

4.3 Semantic Relationships Between Structures and Quotient Structures

Definition 12. *Let G be a group of state mappings of $\mathcal{M}$. A CTL formula φ is G-**invariant** over $\mathcal{M}$, if for every state s, every $g \in G$, for all maximal propositional subformulae φ' of φ, we have*

$$\mathcal{M}, s \models \varphi' \iff \mathcal{M}, g(s) \models \varphi'.$$

Lemma 7. *If φ is G-invariant, then the valuation of φ in $\overline{\mathcal{M}}$ does not depend on the choice of representative map ϑ_G.*

This allows us to connect semantic statements about $\mathcal{M}$ with semantic statements about $\overline{\mathcal{M}}$ for formulae that are G-invariant. The path correspondence theorem establishes a bisimulation between $\mathcal{M}$ and $\overline{\mathcal{M}}$, in which state s of $\mathcal{M}$ and state $\overline{s}$ of $\overline{\mathcal{M}}$ are bisimilar iff s is in the orbit of $\overline{s}$, i.e., $s = g(\overline{s})$ for some $g \in G$. We call such a bisimulation a G-bisimulation. Hence, G-bisimilar states satisfy the same propositional subformulae of any G-invariant CTL formula φ. A straightforward induction over path length then shows that s and $\overline{s}$ satisfy the same G-invariant CTL formulae:

Corollary 1. *$\mathcal{M} \models \varphi$ iff $\overline{\mathcal{M}} \models \varphi$ for all G invariant CTL formulae φ.*

Lemma 8. *Let $s \in \mathcal{M}_S$, $t \in \overline{\mathcal{M}}_S$. Let φ be a G-invariant CTL formula. If $t = \vartheta_G(s)$, then $\mathcal{M}, s \models \varphi \iff \overline{\mathcal{M}}, t \models \varphi$.*

Section 3 developed the theory of substructures of a Kripke structure. This development was motivated by the following definition and theorem.

Definition 13 (Substructure-Repair). *Given a structure $\mathcal{M}$ and a CTL formula φ, we call a nonempty substructure $\mathcal{N}$ of $\mathcal{M}$ a **substructure-repair** of $\mathcal{M}$ with respect to φ if $\mathcal{N} \models \varphi$.*

If a CTL formula φ is G-invariant, then the lattice correspondence will respect the valuation of φ.

Theorem 3 (Repair Correspondence). *Let φ be a G-invariant CTL formula. Let $\mathcal{N}$ be a non-empty G-closed substructure of $\mathcal{M}$, $s \in \mathcal{N}_S$, and $\overline{\mathcal{N}} = \mathcal{N}/(G, \vartheta_G)$. Then $\mathcal{N}, s \models \varphi \iff \overline{\mathcal{N}}, \vartheta_G(s) \models \varphi$.*

5 Repair of Concurrent Programs

A concurrent program $P = P_1 \parallel \ldots \parallel P_n$ consists of n sequential processes executing in parallel. Each process P_i is a set of *i-actions* (s_i, B, t_i), where s_i, t_i are local states of P_i and B is a guard (a predicate on the global state). We say *action* when we ignore the process id. We assume a given set S_0 of initial states. The program $P_1 \parallel \cdots \parallel P_n$ generates a transition $s \xrightarrow{i} t$ iff P_i contains an action (s_i, B, t_i) such that $s{\restriction}i = s_i$, $t{\restriction}i = t_i$, and $s(B) = true$, where $s(B)$ is the value of guard B in global state s. The transition updates only atomic propositions in AP_i, and so $s{\downarrow}i = t{\downarrow}i$. The state-transition graph of P is the closure of this "transition generation" operation, starting in the initial state set S_0.

Given a concurrent program P and a CTL formula φ, we wish to modify P to produce a repaired program P^r such that $\mathcal{M}' \models \varphi$, where $\mathcal{M}'$ is the state-transition graph of P^r. The modification is "subtractive", that is, it only removes behaviors and does not add them. We assume henceforth that when $\mathcal{M}$ is a multiprocess Kripke structure over process indices $1, \ldots, n$, that the symmetry group G is a subgroup of S_n, the group of permutations on $\{1, \ldots, n\}$.

5.1 Repair of Symmetry-reduced Structures

We first generate the symmetry-reduced state transition graph $\overline{\mathcal{M}}$ of P. We use the algorithm of Emerson and Sistla [18, Figure 1]. We then apply the model repair algorithm of Attie et. al. [2] to $\overline{\mathcal{M}}$, and the specification φ of P. This algorithm is sound and complete, so that if $\overline{\mathcal{M}}$ has some substructure that satisfies φ, then the algorithm will return such a substructure $\overline{\mathcal{N}}$. If not, the algorithm will report that no repair exists. As noted, applying this algorithm to the symmetry-reduced state transition graph is only complete with respect to the symmetric repairs, see Example 6.3.

5.2 Extraction of Concurrent Programs from Symmetry-reduced Structures

We want to extract a repaired concurrent program from $\overline{\mathcal{N}}$ using the projection method of [4,13]: each transition $s \xrightarrow{i} t$ is turned into an i-action $action(s \xrightarrow{i} t) \triangleq (s{\restriction}i, B, t{\restriction}i)$, with guard $B = \{\!|s|\!\}$ where $\{\!|s|\!\} \triangleq$ "$(\bigwedge_{Q \in \mathcal{N}_L(s)} Q) \wedge (\bigwedge_{Q \notin \mathcal{N}_L(s)} \neg Q)$" and Q ranges over AP. When process i is in local state s_i, guard B checks that the current global state is actually s.

A key problem is that the definition of the quotient $\overline{\mathcal{M}}$ allows transitions in which the atomic propositions of more than one process are changed, since any representative of an orbit can be chosen. Hence the repaired $\overline{\mathcal{N}} \leq \overline{\mathcal{M}}$ can also contain such transitions, e.g., the transition from **S6** to **S1** in Figure 6 below, which we write as $[C_1\, T_2] \rightarrow [T_1\, N_2]$. Note that the propositions of both processes 1 and 2 are changed. To generate i-actions, such transitions must be converted so that only the atomic propositions of a single process are modified.

Define a transition from s to t to be *regular* iff it modifies atomic propositions in at most one AP_i, so that $s{\downarrow}i = t{\downarrow}i$ for some process index i, and write the

transition as $s \xrightarrow{i} t$. Also define a transition from s to t to be *irregular* iff it is not regular, i.e., it modifies atomic propositions in more than one AP_i, and write the transition as $s \to t$, with no process index labelling the arrow.

For each irregular transition $s \to t \in \overline{\mathcal{N}}_T$, there is $g' \in G$ such that $s \to g'(t)$ is regular. Such an element g' always exists. Let $\overline{s} \to \overline{t} \in \overline{\mathcal{M}}_T$ for arbitrary $\overline{\mathcal{M}}_T$. By Definition 9, there exists $s \to t \in \mathcal{M}_T$ such that $\overline{s} = \vartheta_G(s)$ and $\overline{t} = \vartheta_G(t)$. Hence there is some $g \in G$ such that $g(s) = \overline{s}$ since s and $\overline{s}$ are in the same orbit. Since g is a symmetry of $\mathcal{M}$, we have $g(s) \to g(t) \in \mathcal{M}_T$. Hence $\overline{s} \to g(t) \in \mathcal{M}_T$. Now $t = h(\overline{t})$ for some $h \in G$ since t and $\overline{t}$ are in the same orbit. Hence $\overline{s} \to g(h(\overline{t})) \in \mathcal{M}_T$, and so the needed g' is the product of g and h. For example, by applying the permutation of process indices 1, 2 to $[T_1\, N_2]$, from the irregular transition $[C_1\, T_2] \to [T_1\, N_2]$ we extract the regular transition $[C_1\, T_2] \xrightarrow{1} [N_1\, T_2]$.

Define $Reg_i(\overline{\mathcal{N}}_T)$ to be the set of regular transitions $s \xrightarrow{i} g(t)$ such that $g \in G$ and $s \to t \in \overline{\mathcal{N}}_T$. Since g can be the identity element of G, it follows that this account for both regular and irregular transitions in $\overline{\mathcal{N}}_T$. Define $Act_i(\overline{\mathcal{N}}_T) = \{action(s \xrightarrow{i} t) \mid s \xrightarrow{i} t \in Reg_i(\overline{\mathcal{N}}_T)\}$, be the set of actions obtained from $Reg_i(\overline{\mathcal{N}}_T)$.

Define the action of $g \in G$ on syntactic elements of P_i as follows. For local state s_i: $g(s_i) = s_{g(i)}$. For atomic proposition Q_i: $g(Q_i) = Q_{g(i)}$. For guard B, by induction: $g(\neg B) = \neg g(B)$ and $g(B1 \wedge B2) = g(B1) \wedge g(B2)$, with the base case given by $g(Q_i)$ above. For i-action (s_i, B, t_i): $g(s_i, B, t_i) = (g(s_i), g(B), g(t_i))$. That is, we apply g to all process indices in the syntactic element. Now define $Act_i^G(\overline{\mathcal{N}}_T)$, the symmetrization of $Act_i(\overline{\mathcal{N}}_T)$, by $Act_i^G(\overline{\mathcal{N}}_T) = \{g(a) \mid g \in G, a \in Act_j(\overline{\mathcal{N}}_T), g(j) = i\}$. The repaired concurrent program arises from process-wise repair $\overline{P}^G = \overline{P}_1^G \| \cdots \| \overline{P}_n^G$, where $\overline{P}_i^G$ consists of the i-actions in $Act_i^G(\overline{\mathcal{N}}_T)$.

Theorem 4. *Let $\overline{P}^G$ be the concurrent program extracted from $\overline{\mathcal{N}}$ as above, let $\mathcal{N}^p$ be the state transition graph generated by the execution of $\overline{P}^G$, and let $\overline{\mathcal{N}^p} = \mathcal{N}^p/(G, \vartheta_G)$. Then $\mathcal{N}^p$ is G-closed and $\overline{\mathcal{N}^p} = \overline{\mathcal{N}}$.*

Corollary 2. *Let $\overline{P}^G$ be the repaired program and φ the CTL specification that was used to repair $\overline{\mathcal{M}}$, resulting in $\overline{\mathcal{N}}$. Then $\overline{P}^G \models \varphi$.*

6 Examples

6.1 Two process Mutual Exclusion

We consider mutual exclusion for two processes P_1, P_2. Each P_i has three local states: N_i (neutral, computing locally), T_i (trying, has requested critical section entry), and C_i (in the critical region). We start with the "trivial" program P shown in Figure 4 in which all action guards are "true" and apply the program repair algorithm of Section 5 to repair this program w.r.t. the specification $\varphi = \mathsf{AG}\neg(C_1 \wedge C_2) \wedge \mathsf{AG}((T_1 \vee T_2) \Rightarrow \mathsf{AF}(C_1 \vee C_2))$. The first conjunct specifies mutual

exclusion of the critical sections (safety) and the second specifies progress: if some process requests the critical section then some process will obtain it (liveness). Figure 5 (left side) shows the Kripke structure $\mathcal{M}$ generated by execution of P. Transitions of P_1, P_2 are shown in blue, red, respectively. Clearly, $\mathcal{M} \not\models \varphi$. Actually both conjuncts are violated: $\mathsf{AG}\neg(C_1 \wedge C_2)$ due to the reachability of state **S8** from the initial state, and $\mathsf{AG}((T_1 \vee T_2) \Rightarrow \mathsf{AF}(C_1 \vee C_2))$ due to the self loop on state **S4**.

$\mathcal{M}$ has exactly two symmetries: the identity map, and the map that swaps process indices 1 and 2. Our program repair algorithm does not generate $\mathcal{M}$ since $\mathcal{M}$ may be large, and we show $\mathcal{M}$ only for exposition. We generate $\overline{\mathcal{M}} = \mathcal{M}/(G, \vartheta_G)$ directly from P, and we show $\overline{\mathcal{M}}$ in Figure 5 (right side). $\overline{\mathcal{M}}$ has a transition (shown in black) from state **S6** to **S1**, which is the quotient of the transition from **S6** to **S2** in $\mathcal{M}$, i.e., $\vartheta_G(\mathbf{S6}) = \mathbf{S6}$ and $\vartheta_G(\mathbf{S2}) = \mathbf{S1}$ so the edge $(\vartheta_G(\mathbf{S6}), \vartheta_G(\mathbf{S2}))$ occurs in $\overline{\mathcal{M}}$.

Figure 6 shows the repair $\overline{\mathcal{N}}$ of the reduced structure $\overline{\mathcal{M}}$, and the resultant lifting of the repair to $\mathcal{M}$. The deleted transitions and states are shown dashed. Figure 7 shows the repaired concurrent program $\overline{P}^G$ that is extracted from $\overline{\mathcal{N}}$. Note that $\oplus$ means disjunction [3]. By Corollary 2, $\overline{P}^G \models \varphi$.

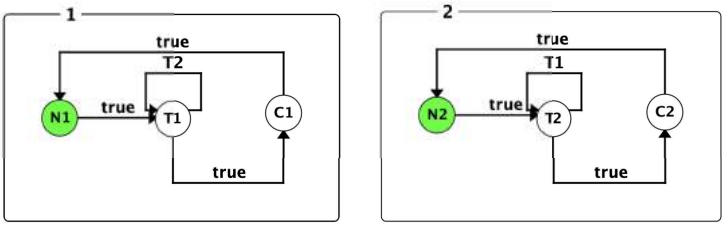


Fig. 4. Initial incorrect mutual exclusion program from Section 6.1.

6.2 *n*-Process Mutual Exclusion

We now consider mutual exclusion for n-processes. To reduce clutter, we remove the trying **Ti** state, and we give a concrete example for 3 processes — the generalization to n processes is straightforward. Each process can move directly from N to C with the appropriate indexes, i.e., the guards on all actions are initially "true", just like in Figure 4.

We consider the mutual exclusion specification $\bigwedge_{i \neq j} \mathsf{AG}\neg(C_i \wedge C_j)$. The group of state mappings G for both structure and specification is the full permutation group on the indices $\{1, \ldots, N\}$. For N-processes, we have that the quotient model by the full group of symmetries has $N+1$ states, while the original model would have 2^N states. Figure 8 shows the repair of the quotient $\overline{\mathcal{M}}$ and then

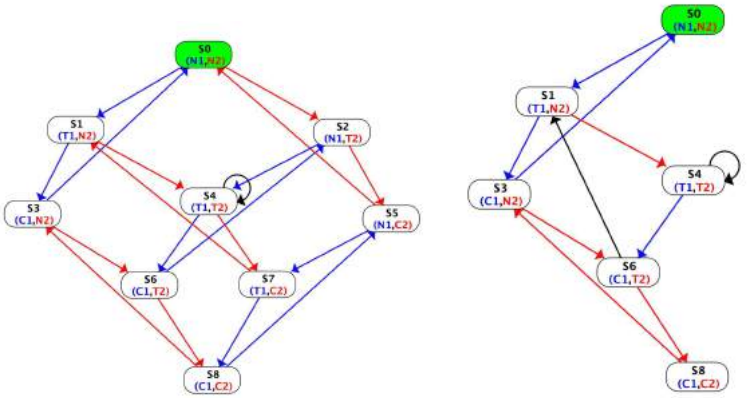


Fig. 5. The original model $\mathcal{M}$ and quotient $\overline{\mathcal{M}} = \mathcal{M}/(G, \vartheta_G)$ for the Kripke structures in Section 6.1.

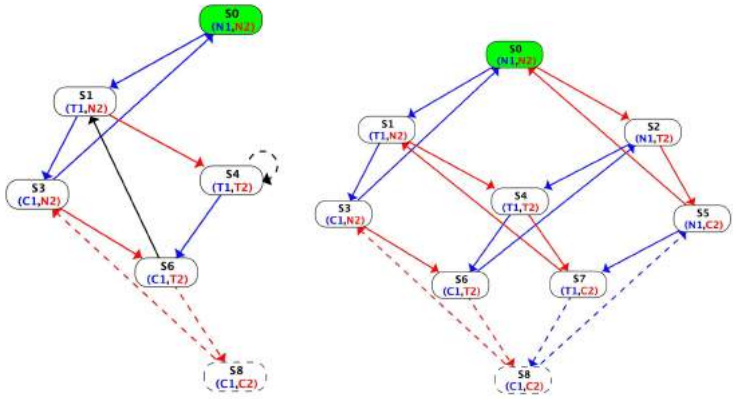


Fig. 6. The repair of $\overline{\mathcal{M}}$ and the lifting of the repair to $\mathcal{M}$ from Section 6.1.

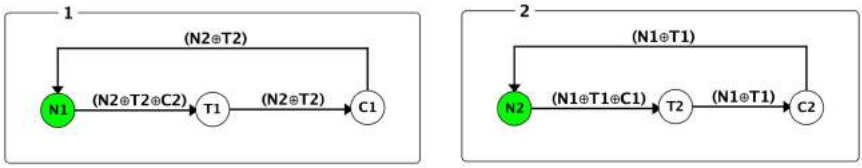


Fig. 7. The mutual exclusion concurrent program extracted from $\mathcal{M}$ in Figure 6.

the lifting of the repair to the original structure $\mathcal{M}$. Figure 9 shows the correct (repaired) program $\overline{P}^G$ that is extracted from the repaired quotient in Figure 8. For N processes, the guard on actions of $\overline{P}_i^G$ is $\bigwedge_{j \neq i} N_j$.

6.3 No G-closed Repairs

Consider the structure in Figure 10 and the formula $f = \mathsf{AXAXAX}P$. The structure $\mathcal{M}$ has a single initial state. Let G be the group consisting of the identity and the map swapping $S1$ and $S2$. In Figure 10 we see that the quotient structure $\mathcal{M}/(G, \vartheta_G)$ does not have any nonempty repairs with respect to f. But, $\mathcal{M}$ does contain a substructure $\mathcal{N}$ that satisfies f.

7 Relative Completeness of Group Theoretic Repair

By the Repair Correspondence (Theorem 3), the existence of a repair $\overline{\mathcal{N}}$ of $\overline{\mathcal{M}}$ implies the existence of a repair $\mathcal{N}$ of $\mathcal{M}$. In Example 6.3, we gave an example in which a repair $\mathcal{N}$ of $\mathcal{M}$ exists but no G-closed repair does, i.e., $\overline{\mathcal{M}}$ has no repairs. This leads us to ask: is there a fragment of CTL, and/or a class of Kripke structures, for which group theoretic repair is complete? That is, the existence of a repair (substructure $\mathcal{N}$ of $\mathcal{M}$ that satisfies φ) implies the existence of a G-closed repair (substructure $\overline{\mathcal{N}}$ of $\overline{\mathcal{M}}$ that satisfies φ).

One attempt to answer this question is to examine formulae and structures where substructures are equivalent to the smallest G-closed substructure containing them. Assume there exists $\mathcal{N} \leq \mathcal{M}$ such that $\mathcal{N} \models \varphi$. Write $\mathcal{N}^G$ for the smallest G-closed structure that contains $\mathcal{N}$. We call $\mathcal{N}^G$ the G-closure of $\mathcal{N}$ in $\mathcal{M}$. If $\mathcal{N}^G$ is bisimilar to $\mathcal{N}$, then $\mathcal{N}^G \models \varphi$ and $\overline{\mathcal{N}^G} \models \varphi$ which is a substructure of $\overline{\mathcal{M}}$.

In [14], Emerson et al., give a criteria for a structure $\mathcal{M}$ to be bisimilar to the symmetrized structure $\mathcal{M}^G$, their criteria is: for any transition $(s,t) \in \left(\mathcal{M}^G\right)_T$, there must be a $g \in G$ such that $(s, gt) \in \mathcal{M}_T$. When asking about substructures, it is not clear what criteria on $\mathcal{M}$ is needed to ensure that each substructure $\mathcal{N}$ of $\mathcal{M}$ is bisimilar to $\mathcal{N}^G$.

Definition 14 (G-Repair Complete). *Let $\mathcal{M}$ be a Kripke structure with a group of state mappings G and φ a G-invariant* CTL *formula. Let $\mathcal{N} \leq \mathcal{M}$ be any repair of $\mathcal{M}$ with respect to φ, and let s be any state in $\mathcal{N}_S$. Then the pair $(\mathcal{M}, \varphi)$ is G-repair complete if: $\mathcal{N}, s \models \varphi$ implies for all $g \in G$, we have $\mathcal{N}^G, g(s) \models \varphi$.*

It is clear that propositional formulae are always G-repair complete. In addition we note the following:

Theorem 5. *If φ and ψ are purely propositional formulae then for any Kripke structure $\mathcal{M}$, the pair $(\mathcal{M}, \mathsf{A}[\varphi \,\mathsf{R}\, \psi])$ is G-repair complete.*

There exists structures $\mathcal{M}$ and φ, ψ formulae such that $(\mathcal{M}, \varphi)$ G-repair complete, and $(\mathcal{M}, \psi)$ G-repair complete, but $(\mathcal{M}, \varphi \wedge \psi)$ not G-repair complete.

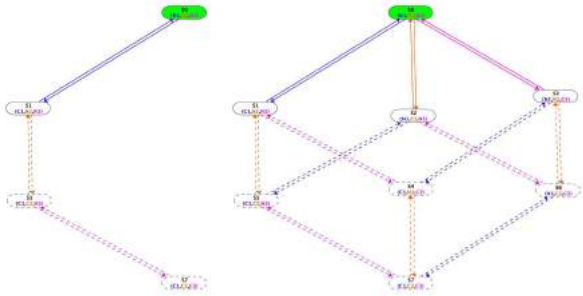


Fig. 8. The Kripke structure defined in Section 6.2. On the left is the repair of $\overline{\mathcal{M}}$ and the lifting of the repair to $\mathcal{M}$ appears on the right.

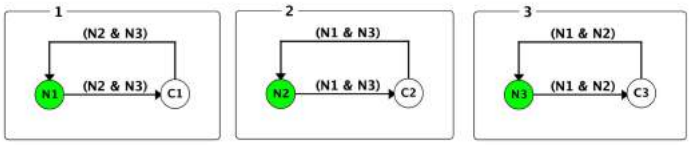


Fig. 9. The repaired program $\overline{P}^G$ for the program in Section 6.2.

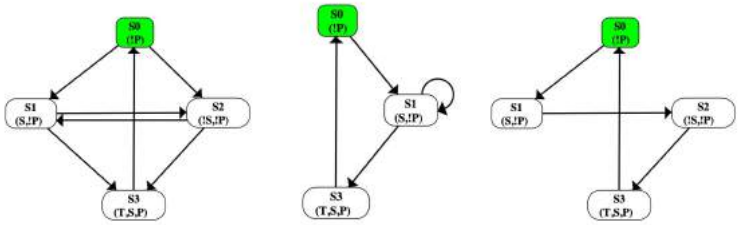


Fig. 10. The models from Section 6.3 from left to right: the model $\mathcal{M}$, the quotient of $\mathcal{M}$, a repair of $\mathcal{M}$ with respect to $f = \mathsf{AXAXAX}P$ that is not G-closed.

Example 4. Let $\mathcal{M}$ be the Kripke structure described by Figure 11. Let G be the group of state mappings generated by swapping $s1$ and $s2$. Let $\varphi = \mathsf{A}[p\,\mathsf{R}\,q]$ and $\psi = \mathsf{AF} \neq q$. The structure $\mathcal{M}$ has a nonempty G-closed repair for φ. Similarly there is a single nonempty G-closed repair for ψ. But $\mathcal{M}$ has no G-closed repairs of $\varphi \wedge \psi$.

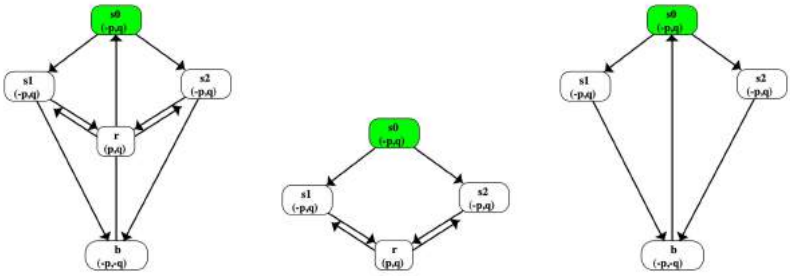


Fig. 11. The Kripke Structure from Example 4 (note that (b, s_0) is a transition, while (b, r) is not) (left), G-closed repairs of $\mathcal{M}$ with respect to the formulae $\mathsf{A}[p\,\mathsf{R}\,q]$ (center), and $\mathsf{AF}\neg q$ (right).

8 Conclusions

We present a theory of how group actions could be used to assist in the repair of a Kripke structure.

We presented a theory for the substructures of a given Kripke structure $\mathcal{M}$, their organization into lattices, and how these substructures interact with a group of state-mappings of $\mathcal{M}$. We show a lattice isomorphism between substructure-repairs of $\overline{\mathcal{M}}$ and G-maximal repairs of $\mathcal{M}$ (Theorem 3: Repair Correspondence). This monotone Galois correspondence guarantees that a repair of $\overline{\mathcal{M}}$ lifts to a repair of $\mathcal{M}$: that is to say that model repairs of $\overline{\mathcal{M}}$ with respect to a G-invariant CTL formula φ lift to model repairs of $\mathcal{M}$ with respect to φ. Using this theory we were able to devise a method for repairing concurrent programs which exploits this correspondence, thus avoiding state explosion. We construct the quotient structure $\overline{\mathcal{M}}$ directly from P without the need to construct the structure $\mathcal{M}$. By our correspondence, repairing $\overline{\mathcal{M}}$ will lift to a repair of the structure $\mathcal{M}$, which in turn corresponds to a repair of P. We show how to construct a repair of P using the repair of $\overline{\mathcal{M}}$ while circumventing the creation of the larger Kripke structure.

A Kripke structure $\mathcal{M}$ that can be repaired with respect to a formulae φ can be repaired via abstraction. However, not every repair of an abstracted structure $\mathcal{N}$ corresponds to a repair of $\mathcal{M}$. In contrast, the structure might not be repairable using the quotient structure, but any repair of the quotient structure will lift to a repair of the original structure.

References

1. Aminof, B., Jacobs, S., Khalimov, A., Rubin, S.: Parameterized model checking of token-passing systems. In: McMillan, K.L., Rival, X. (eds.) Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8318, pp. 262–281. Springer (2014). https://doi.org/10.1007/978-3-642-54013-4_15, https://doi.org/10.1007/978-3-642-54013-4_15
2. Attie, P.C., Dak-Al-Bab, K., Sakr, M.: Model and program repair via SAT solving. ACM Trans. Embed. Comput. Syst. **17**(2), 32:1–32:25 (2018). https://doi.org/10.1145/3147426, https://doi.org/10.1145/3147426
3. Attie, P.C., Emerson, E.A.: Synthesis of concurrent systems with many similar processes. ACM Trans. Program. Lang. Syst. **20**(1), 51–115 (jan 1998). https://doi.org/10.1145/271510.271519, https://doi.org/10.1145/271510.271519
4. Attie, P.C., Emerson, E.A.: Synthesis of concurrent programs for an atomic read/write model of computation. ACM Trans. Program. Lang. Syst. **23**(2), 187–242 (mar 2001). https://doi.org/10.1145/383043.383044, https://doi.org/10.1145/383043.383044
5. Buccafurri, F., Eiter, T., Gottlob, G., Leone, N.: Enhancing model checking in verification by AI techniques. Artif. Intell. **112**, 57–104 (1999)
6. Chatzieleftheriou, G., Bonakdarpour, B., Smolka, S., Katsaros, P.: Abstract model repair. In: Goodloe, A., Person, S. (eds.) NASA Formal Methods, Lecture Notes in Computer Science, vol. 7226, pp. 341–355. Springer Berlin Heidelberg, Norfolk, VA, USA (2012)
7. Clarke, E.M., Emerson, E.A., Jha, S., Sistla, A.P.: Symmetry reductions in model checking. In: CAV. Lecture Notes in Computer Science, vol. 1427, pp. 147–158. Springer (1998)
8. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. **8**(2), 244–263 (1986)
9. Clarke, E.M., Grumberg, O., Jha, S.: Verifying parameterized networks. ACM Trans. Program. Lang. Syst. **19**(5), 726–750 (1997). https://doi.org/10.1145/265943.265960, https://doi.org/10.1145/265943.265960
10. Clarke, E.M., Jha, S., Enders, R., Filkorn, T.: Exploiting symmetry in temporal logic model checking. Formal Methods Syst. Des. **9**(1/2), 77–104 (1996). https://doi.org/10.1007/BF00625969, https://doi.org/10.1007/BF00625969
11. Donaldson, A.F., Miller, A.: Automatic symmetry detection for model checking using computational group theory. In: International Symposium on Formal Methods. pp. 481–496. Springer (2005)
12. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), pp. 995–1072. Elsevier and MIT Press (1990)
13. Emerson, E.A., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Sci. Comput. Program. **2**(3), 241–266 (1982)
14. Emerson, E.A., Havlicek, J., Trefler, R.J.: Virtual symmetry reduction. In: LICS. pp. 121–131. IEEE Computer Society (2000)
15. Emerson, E.A., Kahlon, V.: Parameterized model checking of ring-based message passing systems. In: CSL. Lecture Notes in Computer Science, vol. 3210, pp. 325–339. Springer (2004)

16. Emerson, E.A., Namjoshi, K.S.: Reasoning about rings. In: POPL. pp. 85–94. ACM Press (1995)
17. Emerson, E.A., Namjoshi, K.S.: On reasoning about rings. Int. J. Found. Comput. Sci. **14**(4), 527–550 (2003)
18. Emerson, E.A., Sistla, A.P.: Symmetry and model checking. Formal Methods Syst. Des. **9**(1/2), 105–131 (1996)
19. Emerson, E.A., Sistla, A.P.: Utilizing symmetry when model-checking under fairness assumptions: An automata-theoretic approach. ACM Trans. Program. Lang. Syst. **19**(4), 617–638 (1997)
20. Emerson, E.A., Trefler, R.J.: Model checking real-time properties of symmetric systems. In: MFCS. Lecture Notes in Computer Science, vol. 1450, pp. 427–436. Springer (1998)
21. Emerson, E.A., Trefler, R.J.: From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In: CHARME. Lecture Notes in Computer Science, vol. 1703, pp. 142–156. Springer (1999)
22. Emerson, E.A., Wahl, T.: Dynamic symmetry reduction. In: TACAS. Lecture Notes in Computer Science, vol. 3440, pp. 382–396. Springer (2005)
23. von Essen, C., Jobstmann, B.: Program repair without regret. Formal Methods Syst. Des. **47**(1), 26–50 (2015). https://doi.org/10.1007/s10703-015-0223-6, https://doi.org/10.1007/s10703-015-0223-6
24. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM **39**(3), 675–735 (1992). https://doi.org/10.1145/146637.146681, https://doi.org/10.1145/146637.146681
25. Goues, C.L., Pradel, M., Roychoudhury, A.: Automated program repair. Commun. ACM **62**(12), 56–65 (nov 2019). https://doi.org/10.1145/3318162, https://doi.org/10.1145/3318162
26. Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Trans. Program. Lang. Syst. **16**(3), 843–871 (may 1994). https://doi.org/10.1145/177492.177725, https://doi.org/10.1145/177492.177725
27. Isaacs, I.M.: Finite group theory, vol. 92. American Mathematical Soc. (2008)
28. Isaacs, I.M.: Algebra: a graduate course, vol. 100. American Mathematical Soc. (2009)
29. Jobstmann, B., Griesmayer, A., Bloem, R.: Program repair as a game. In: CAV. pp. 226–238. Springer-Verlag, Berlin, Heidelberg (2005)
30. Namjoshi, K.S., Trefler, R.J.: Uncovering symmetries in irregular process networks. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 496–514. Springer (2013)
31. Serre, J.P.: Finite groups: an introduction. International Press Somerville, MA (2016)
32. Staber, S., Jobstmann, B., Bloem, R.: Finding and fixing faults. In: CHARME '05. pp. 35–49. Springer-Verlag, Berlin, Heidelberg (2005), springer LNCS no. 3725

Subgame Optimal Strategies in Finite Concurrent Games with Prefix-Independent Objectives

Benjamin Bordais(✉), Patricia Bouyer and Stéphane Le Roux

Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190 Gif-sur-Yvette, France
bordais@lsv.fr

Abstract. We investigate concurrent two-player win/lose stochastic games on finite graphs with prefix-independent objectives. We characterize subgame optimal strategies and use this characterization to show various memory transfer results: 1) For a given (prefix-independent) objective, if every game that has a subgame *almost-surely winning* strategy also has a positional one, then every game that has a subgame *optimal* strategy also has a positional one; 2) Assume that the (prefix-independent) objective has a neutral color. If every *turn-based* game that has a subgame almost-surely winning strategy also has a positional one, then every game that has a *finite-choice* (notion to be defined) subgame optimal strategy also has a positional one.
We collect or design examples to show that our results are tight in several ways. We also apply our results to Büchi, co-Büchi, parity, mean-payoff objectives, thus yielding simpler statements.

1 Introduction

Turn-based two-player win/lose (stochastic) games on finite graphs have been intensively studied in the context of model checking in a broad sense [19,1]. These games behave well regarding optimality in various settings. Most importantly for this paper, [14] proved the following results for finite turn-based stochastic games with prefix-independent objectives: (1) every game has deterministic optimal strategies; (2) from every value-1 state, there is an optimal, i.e. almost-surely winning, strategy; (3) if from every value-1 state of every game there is an optimal strategy using some fixed amount of memory, every game has an optimal strategy using this amount of memory. These results are of either of the following generic forms:

- In all games, (from all nice states) there is a nice strategy.
- If from all *nice states* of all games there is a nice strategy, so it is from all *states*.

The concurrent version of these turn-based (stochastic) games has a higher modeling power than the turn-based version: this is really useful in practice since real-world systems are intrinsically concurrent [15]. They are played on a finite graph as follows: at each player state, the two players stochastically and independently choose one among finitely many actions. This yields a Nature state,

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 541–560, 2023.
https://doi.org/10.1007/978-3-031-30829-1_26

which stochastically draws a next player state, from where each player chooses one action again, and so on. Each player state is labelled by a color, and who wins depends on the infinite sequence of colors underlying the (stochastically) generated infinite sequence of player states. Unfortunately, these concurrent games do not behave well in general even for simple winning conditions and simple graph structures, like finite graphs:

- Reachability objectives: there is a game without optimal strategies [13];
- Büchi objectives: there is a game with value 1 while all finite-memory strategies have value 0 [12];
- Co-Büchi objectives: although there are always positional ε-optimal strategies [8], there is a game with optimal strategies but without finite-memory optimal strategies [4];
- Parity [12] and mean-payoff [10] objectives: there is a game with subgame almost-surely-winning strategies, but where all finite-memory strategies have value 0.

In this paper, we focus on concurrent stochastic finite games. Therefore, the generic forms of our results will be more complex, in order to take into account the above-mentioned discrepancies. They will somehow be given as generic statements as follows:

- Every game that has a *nice* strategy also has a *nicer* one.
- If all *special games* that have a nice strategy have a nicer one, so it is for all *games*.

Much of the difficulty consists in fine-tuning the strength of "nice", "nicer" and "special" above. We present below our main contributions on finite two-player win/lose concurrent stochastic games with prefix-independent objectives:

1. We provide a characterization of subgame optimal strategies, which are strategies that are optimal after every history (Theorem 1): a Player A strategy is subgame optimal iff 1) it is locally optimal and 2) for every Player B deterministic strategy, after every history, if the visited states have the same positive value, Player A wins with probability 1. This characterization is used to prove all the results below.
2. We prove memory transfer results from subgame almost-surely winning strategies to subgame optimal strategies:
   (a) Theorem 2: If every game that has a subgame *almost-surely winning* strategy also has a positional one, then every game that has a subgame *optimal* strategy also has a positional one.
   (b) Corollary 1: every Büchi or co-Büchi game that has a subgame optimal strategy has a positional one. (Whereas parity games may require infinite memory [12].)

   Note that the transfer result **2a** can be generalized from positional to finite memory.
3. We say that a strategy has finite-choice, if it uses only finitely many action distributions. Note that finite-memory (resp. deterministic) strategies clearly have finite choice.

(a) Theorem 4: In a given game, if there is a finite-choice optimal strategy, there is a finite-choice *subgame* optimal strategy.
(b) Theorem 5: Assume that the objective has a neutral color. If every *turn-based* game that has a subgame almost-surely winning strategy also has a positional one, then every game that has a *finite-choice* subgame optimal strategy also has a positional one.
(c) Corollary 2: every parity or mean-payoff game that has a finite-memory subgame optimal strategy also has a positional one.

Note that **3a** and **3b** are false if the word finite-choice is removed [4]. The proof of **3b** invokes **3a**. Flavor (and proofs) of **3b** and **2a** are similar, but both premises and conclusions are weakened in **3b**, as emphasized.

Related works. A large part of this paper is dedicated to the extension to concurrent games of the results from [14] regarding the transfer of memory from almost-surely winning strategies to optimal strategies in turn-based games. Note that the proof technique used in [14] is different and could not be adapted to our more general setting. In their proof, both players agree on a preference over Nature states and play according to this preference. In our proof, we slice the graph into value areas (that is, sets of states with the same value), and show that it is sufficient to play an almost-sure winning strategy in each slice; we then glue these (partial) strategies together to get a subgame-optimal strategy over the whole graph.

The slicing technique was already used in the context of concurrent games in [8]. The authors focus on parity objectives and establishes a memory transfer result from limit-sure winning strategies to almost-optimal strategies. As an application, they show that, for co-Büchi objectives, since positional strategies are sufficient to win limit-surely, they also are to win almost-optimally. Their construction made heavy use of the specific nature of parity objectives.

We also mention [6], where the focus is also on concurrent games with prefix-independent objectives. In particular, the authors establish a (very useful) result: if all states have positive values, then they all have value 1. (Note that a strengthening of this result is presented in this paper (Theorem 3), which also appears as an adaptation of a result proved in [14]). This result is then used in another context with non-zero-sum games.

Finally, some recent works on concurrent games have been done in [2,3,4], where the goal is the following: local interactions of the two players in the player state are given by bi-dimensional tables; those tables can be abstracted as *game forms*, where (output) variables are issues of the local interaction (possibly several issues are labelled by the same variable). The goal of this series of works is to give (intrinsic) properties of these game forms, so that, when used in a graph game, the existence of optimal strategies is ensured. For instance, in [3], a property of games forms, called RM, is given, which ensures that, if one only uses RM game forms in a graph, then for every reachability objective, Player A will always have an optimal strategy for that objective. This property is a characterization of well-behaved game forms regarding reachability objectives

since every game form which is not RM can be embedded into a (small) graph game in such a way that Player A does not have an optimal strategy. This line of works really differs from the target of the current paper.

Structure of the paper. Section 2 presents notations, Section 3 recalls the notion of game forms, Section 4 introduces our formalism, Section 5 exhibits a necessary and sufficient pair of conditions for subgame optimality, Section 6 shows a memory transfer from subgame almost-surely winning to subgame optimal in concurrent games, and Section 7 adapts the results of the previous section to the case of the existence of a subgame finite-choice strategy.

Detailed proofs and additional formal definitions are available in [5].

2 Preliminaries

Consider a non-empty set Q. We denote by Q^*, Q^+ and Q^ω the set of finite sequences, non-empty finite sequences and infinite sequences of elements of Q respectively. For $n \in \mathbb{N}$, we denote by Q^n (resp. $Q^{\leq n}$) the set of sequences of (resp. at most) n elements of Q. For all $\rho = q_1 \cdots q_n \in Q^n$ and $i \leq n$, we denote by ρ_i the element $q_i \in Q$ and by $\rho_{\leq i} \in Q^i$ the finite sequence $q_1 \cdots q_i$. For a subset $S \subseteq Q$, we denote by $Q^* \cdot S^\omega \subseteq Q^\omega$ the set of infinite paths that eventually settle in S and by $(Q^* \cdot S)^\omega \subseteq Q^\omega$ the set of infinite paths visiting infinitely often the set S.

A *discrete probabilistic distribution* over a non-empty finite set Q is a function $\mu : Q \to [0,1]$ such that $\sum_{x \in Q} \mu(x) = 1$. The *support* $\mathsf{Supp}(\mu)$ of a probabilistic distribution $\mu : Q \to [0,1]$ is the set of non-zeros of the distribution: $\mathsf{Supp}(\mu) = \{q \in Q \mid \mu(q) \in (0,1]\}$. The set of all distributions over Q is denoted $\mathcal{D}(Q)$.

3 Game forms

We recall the definition of game forms – informally, bi-dimensional tables with variables – and of games in normal forms – game forms whose outcomes are values between 0 and 1.

Definition 1 (Game form and game in normal form). *A* game form *(GF for short) is a tuple* $\mathcal{F} = \langle \mathsf{Act_A}, \mathsf{Act_B}, \mathsf{O}, \varrho \rangle$ *where* $\mathsf{Act_A}$ *(resp.* $\mathsf{Act_B}$*) is the non-empty finite set of actions available to Player* A *(resp.* B*),* O *is a non-empty set of outcomes, and* $\varrho : \mathsf{Act_A} \times \mathsf{Act_B} \to \mathsf{O}$ *is a function that associates an outcome to each pair of actions. When the set of outcomes* O *is equal to* $[0,1]$*, we say that* $\mathcal{F}$ *is a* game in normal form. *For a valuation* $v \in [0,1]^{\mathsf{O}}$ *of the outcomes, the notation* $\langle \mathcal{F}, v \rangle$ *refers to the game in normal form* $\langle \mathsf{Act_A}, \mathsf{Act_B}, [0,1], v \circ \varrho \rangle$.

We use game forms to represent interactions between two players. The strategies available to Player A (resp. B) are convex combinations of actions given as the rows (resp. columns) of the table. In a game in normal form, Player A tries to maximize the outcome, whereas Player B tries to minimize it.

Definition 2 (Outcome of a game in normal form). *Consider a game in normal form* $\mathcal{F} = \langle \mathsf{Act_A}, \mathsf{Act_B}, [0,1], \varrho \rangle$. *The set* $\mathcal{D}(\mathsf{Act_A})$ *(resp.* $\mathcal{D}(\mathsf{Act_B})$*) is the set of strategies available to Player* A *(resp.* B*). For a pair of strategies* $(\sigma_\mathsf{A}, \sigma_\mathsf{B}) \in \mathcal{D}(\mathsf{Act_A}) \times \mathcal{D}(\mathsf{Act_B})$*, the outcome* $\mathsf{out}_\mathcal{F}(\sigma_\mathsf{A}, \sigma_\mathsf{B})$ *in* $\mathcal{F}$ *of the strategies* $(\sigma_\mathsf{A}, \sigma_\mathsf{B})$ *is defined as:* $\mathsf{out}_\mathcal{F}(\sigma_\mathsf{A}, \sigma_\mathsf{B}) := \sum_{a \in \mathsf{Act_A}} \sum_{b \in \mathsf{Act_B}} \sigma_\mathsf{A}(a) \cdot \sigma_\mathsf{B}(b) \cdot \varrho(a,b) \in [0,1]$.

Definition 3 (Value of a game in normal form and optimal strategies). *Consider a game in normal form* $\mathcal{F} = \langle \mathsf{Act_A}, \mathsf{Act_B}, [0,1], \varrho \rangle$ *and a strategy* $\sigma_\mathsf{A} \in \mathcal{D}(\mathsf{Act_A})$ *for Player* A*. The* value *of the strategy* σ_A*, denoted* $\mathsf{val}_\mathcal{F}(\sigma_\mathsf{A})$ *is equal to:* $\mathsf{val}_\mathcal{F}(\sigma_\mathsf{A}) := \inf_{\sigma_\mathsf{B} \in \mathcal{D}(\mathsf{Act_B})} \mathsf{out}_\mathcal{F}(\sigma_\mathsf{A}, \sigma_\mathsf{B})$*, and analogously for Player* B*, with a* sup *instead of an* inf*. When* $\sup_{\sigma_\mathsf{A} \in \mathcal{D}(\mathsf{Act_A})} \mathsf{val}_\mathcal{F}(\sigma_\mathsf{A}) = \inf_{\sigma_\mathsf{B} \in \mathcal{D}(\mathsf{Act_B})} \mathsf{val}_\mathcal{F}(\sigma_\mathsf{B})$*, it defines the* value *of the game* $\mathcal{F}$*, denoted* $\mathsf{val}_\mathcal{F}$*.*

A strategy $\sigma_\mathsf{A} \in \mathcal{D}(\mathsf{Act_A})$ *ensuring* $\mathsf{val}_\mathcal{F} = \mathsf{val}_\mathcal{F}(\sigma_\mathsf{A})$ *is called* optimal*. The set of all optimal strategies for Player* A *is denoted* $\mathsf{Opt_A}(\mathcal{F}) \subseteq \mathcal{D}(\mathsf{Act_A})$*, and analogously for Player* B*. Von Neuman's minimax theorem [20] ensures the existence of optimal strategies (for both players).*

In the following, strategies in games in normal forms will be called GF-strategies, in order not to confuse them with strategies in concurrent (graph) games.

4 Concurrent games and optimal strategies

4.1 Concurrent arenas and strategies

We introduce the definition of concurrent arenas played on a finite graph.

Definition 4 (Finite stochastic concurrent arena). *A* colored concurrent arena $\mathcal{C}$ *is a tuple* $\langle Q, (A_q)_{q \in Q}, (B_q)_{q \in Q}, \mathsf{D}, \delta, \mathsf{dist}, \mathsf{K}, \mathsf{col} \rangle$ *where* Q *is the non-empty finite set of states, for all* $q \in Q$*,* A_q *(resp.* B_q*) is the non-empty finite set of actions available to Player* A *(resp.* B*) at state* q*,* D *is the finite set of Nature states,* $\delta : \bigcup_{q \in Q} (\{q\} \times A_q \times B_q) \to \mathsf{D}$ *is the transition function,* $\mathsf{dist} : \mathsf{D} \to \mathcal{D}(Q)$ *is the distribution function. Furthermore,* K *is the non-empty finite set of colors and* $\mathsf{col} : Q \to \mathsf{K}$ *is the coloring function.*

In the following, the arena $\mathcal{C}$ will refer to the tuple $\langle Q, (A_q)_{q \in Q}, (B_q)_{q \in Q}, \mathsf{D}, \delta, \mathsf{dist}, \mathsf{K}, \mathsf{col} \rangle$, unless otherwise stated. A concurrent game is obtained from a concurrent arena by adding a winning condition: the set of infinite paths winning for Player A (and losing for Player B).

Definition 5 (Finite stochastic concurrent game). *A finite* concurrent game *is a pair* $\langle \mathcal{C}, W \rangle$ *where* $\mathcal{C}$ *is a finite concurrent colored arena and* $W \subseteq \mathsf{K}^\omega$ *is Borel. The set* W *is called the* objective*, as it corresponds to the set of colored paths winning for Player* A*.*

In this paper, we only consider a specific kind of objectives: prefix-independent ones. Informally, they correspond to objectives W such that an infinite path ρ is in W if and only if any of its suffixes is in W. More formally:

Definition 6 (Prefix-independent objectives). *For a non-empty finite set of colors* K *and* $W \subseteq \mathsf{K}^\omega$*,* W *is said to be* prefix-independent *(PI for short) if, for all* $\rho \in \mathsf{K}^\omega$ *and* $i \geq 0$*,* $\rho \in W \Leftrightarrow \rho_{\geq i} \in W$*.*

In the following, we refer to concurrent games with prefix-independent objectives as PI concurrent games.

Definition 7 (Parity, Büchi, co-Büchi objectives). *Let* $\mathsf{K} \subset \mathbb{N}$ *be a finite non-empty set of integers. Consider a concurrent arena* $\mathcal{C}$ *with* K *as set of colors. For an infinite path* $\rho \in Q^\omega$*, we denote by* $\mathsf{col}(\rho)_\infty \subseteq \mathbb{N}$ *the set of colors seen infinitely often in* ρ*:* $\mathsf{col}(\rho)_\infty := \{n \in \mathbb{N} \mid \forall i \in \mathbb{N},\ \exists j \geq i,\ \mathsf{col}(\rho_j) = n\}$*. Then, the* parity objective *w.r.t.* col *is the set* $W^{\mathsf{Parity}}(\mathsf{col}) := \{\rho \in Q^\omega \mid \max \mathsf{col}(\rho)_\infty$ *is even* $\}$*. The Büchi (resp. co-Büchi) objective correspond to the parity objective with* $\mathsf{K} := \{1, 2\}$ *(resp.* $\mathsf{K} := \{0, 1\}$*).*

Strategies are then defined as functions that, given the history of the game (i.e. the sequence of states already seen) associate a distribution on the actions available to the Player.

Definition 8 (Strategies). *Consider a concurrent game* $\mathcal{C}$*. A strategy for Player* A *is a function* $\mathsf{s}_\mathsf{A} : Q^+ \to \mathcal{D}(A)$ *with* $A := \bigcup_{q \in Q} A_q$ *such that, for all* $\rho = q_0 \cdots q_n \in Q^+$*, we have* $\mathsf{s}_\mathsf{A}(\rho) \in \mathcal{D}(A_{q_n})$*. We denote by* $\mathsf{S}^\mathsf{A}_\mathcal{C}$ *the set of all strategies in arena* $\mathcal{C}$ *for Player* A*. This is analogous for Player* B*.*

Given two strategies $\mathsf{s}_\mathsf{A}, \mathsf{s}_\mathsf{B}$ for both players in an arena $\mathcal{C}$ from a starting state q_0, we define in the usual manner the probability $\mathbb{P}^{\mathcal{C},q_0}_{\mathsf{s}_\mathsf{A},\mathsf{s}_\mathsf{B}}$ of a finite path which induces the probability of an arbitrary Borel subset of infinite paths. Values of strategies and of the game are defined below.

Definition 9 (Value of strategies and of the game). *Let* $\mathcal{G} = \langle \mathcal{C}, W \rangle$ *be a PI concurrent game and consider a strategy* $\mathsf{s}_\mathsf{A} \in \mathsf{S}^\mathsf{A}_\mathcal{C}$ *for Player* A*. The function* $\chi_\mathcal{G}[\mathsf{s}_\mathsf{A}] : Q \to [0, 1]$ *giving the value of the strategy* s_A *is such that, for all* $q_0 \in Q$*, we have* $\chi_\mathcal{G}[\mathsf{s}_\mathsf{A}](q_0) := \inf_{\mathsf{s}_\mathsf{B} \in \mathsf{S}^\mathsf{B}_\mathcal{C}} \mathbb{P}^{\mathcal{C},q_0}_{\mathsf{s}_\mathsf{A},\mathsf{s}_\mathsf{B}}[W]$*. The function* $\chi_\mathcal{G}[\mathsf{A}] : Q \to [0, 1]$ *giving the value for Player* A*: is such that, for all* $q_0 \in Q$*, we have* $\chi_\mathcal{G}[\mathsf{A}](q_0) := \sup_{\mathsf{s}_\mathsf{A} \in \mathsf{S}^\mathsf{A}_\mathcal{C}} \chi_\mathcal{G}[\mathsf{s}_\mathsf{A}](q_0)$*. The function* $\chi_\mathcal{G}[\mathsf{B}] : Q \to [0, 1]$ *giving the value of the game for Player* B *is defined similarly by reversing the supremum and infimum.*

By Martin's result on the determinacy of Blackwell games [17], for all concurrent games $\mathcal{G} = \langle \mathcal{C}, W \rangle$*, the value functions for both Players are equal, this defines the value function* $\chi_\mathcal{G} : Q \to [0, 1]$ *of the game:* $\chi_\mathcal{G} := \chi_\mathcal{G}[\mathsf{A}] = \chi_\mathcal{G}[\mathsf{B}]$*.*

We define value areas: subsets of states whose values are the same.

Definition 10 (Value area). *In a PI concurrent game* $\mathcal{G}$*,* $V_\mathcal{G}$ *refers to the set of values appearing in the game:* $V_\mathcal{G} := \{\chi_\mathcal{G}[q] \mid q \in Q\}$*. Furthermore, for all* $u \in V_\mathcal{G}$*,* $Q_u \subseteq Q$ *refers to the set of states whose values are* u *w.r.t.* $\chi_\mathcal{G}$*:* $Q_u := \{q \in Q \mid \chi_\mathcal{G}(q) = u\}$*.*

In concurrent games, game forms appear at each state and describe the interactions of the players at that state. Furthermore, the valuation mapping each

state to its value in the game can be lifted, via a convex combination, into a valuation of the Nature states. This, in turn, induces a natural way to define the game in normal form appearing at each state.

Definition 11 (Local interactions, Lifting valuations). *In a PI concurrent game $\mathcal{G}$ where the valuation $\chi_{\mathcal{G}} : Q \to [0,1]$ gives the values of the game, the lift $\nu_{\mathcal{G}} : \mathsf{D} \to [0,1]$ is such that, for all $d \in \mathsf{D}$, we have $\nu_{\mathcal{G}}(d) := \sum_{q \in Q} \chi_{\mathcal{G}}(q) \cdot \mathsf{dist}(d)(q)$ (recall that $\mathsf{dist} : \mathsf{D} \to \mathcal{D}(Q)$ is the distribution function).*

Let $q \in Q$. The local interaction *at state q is the game form $\mathcal{F}_q = \langle A_q, B_q, \mathsf{D}, \delta(q,\cdot,\cdot)\rangle$. The game in normal form at state q is then $\mathcal{F}_q^{\mathsf{nf}} := \langle \mathcal{F}_q, \nu_{\mathcal{G}}\rangle$.*

The values of the game in normal form $\mathcal{F}_q^{\mathsf{nf}}$ and of the state q are equal.

Proposition 1. *In a PI concurrent game $\mathcal{G}$, for all states $q \in Q$, we have $\chi_{\mathcal{G}}(q) = \mathsf{out}_{\mathcal{F}_q^{\mathsf{nf}}}$.*

4.2 More on strategies

In this subsection, we define several kinds of strategies. Let us fix a PI concurrent game $\mathcal{G}$ for the rest of this section. First, we consider optimal strategies, i.e. strategies realizing the value of the game. Strategies are positively-optimal if their values are positive from all states whose value is positive.

Definition 12 ((Positively-) optimal strategies). *A Player A strategy $\mathsf{s}_\mathsf{A} \in \mathsf{S}^\mathsf{A}_\mathcal{C}$ is (resp. positively-)* optimal *from a state $q \in Q$ if $\chi_{\mathcal{G}}(q) = \chi_{\mathcal{G}}[\mathsf{s}_\mathsf{A}](q)$ (resp. if $\chi_{\mathcal{G}}(q) > 0 \Rightarrow \chi_{\mathcal{G}}[\mathsf{s}_\mathsf{A}](q) > 0$). It is (resp. positively-) optimal if this holds from all states $q \in Q$.*

Note that the definition of optimal strategies we consider is sometimes referred to as uniform optimality, as it holds from every state of the game. However, it does not say anything about what happens once some sequence of states have been seen. We would like now to define a notion of strategy that is optimal from any point that can occur after any finite sequence of states has been seen. This correspond to subgame optimal strategies. To define them, we need to introduce the notion of residual strategy.

Definition 13 (Residual and Subgame Optimal Strategies). *For all finite sequences $\rho \in Q^+$, the* residual strategy $\mathsf{s}^\rho_\mathsf{A}$ *of a Player A strategy s_A is the strategy $\mathsf{s}^\rho_\mathsf{A} : Q^+ \to \mathcal{D}(A)$ such that, for all $\pi \in Q^+$, we have $\mathsf{s}^\rho_\mathsf{A}(\pi) := \mathsf{s}_\mathsf{A}(\rho \cdot \pi)$.*

The Player A strategy s_A is subgame optimal *if, for all $\rho = \rho' \cdot q \in Q^+$, the residual strategy $\mathsf{s}^\rho_\mathsf{A}$ is optimal from q, i.e. $\chi_{\mathcal{G}}[\mathsf{s}^\rho_\mathsf{A}](q) = \chi_{\mathcal{G}}(q)$.*

Note that, in particular, subgame optimal strategies are optimal strategies. When such strategies do exist, we want them to be as simple as possible, for instance we want them to be positional, that is that they only depend on the current state of the game.

As for Player B, we will consider a specific kind of strategies, namely deterministic strategies. That is because, once a Player A strategy is fixed we obtain an (infinite) MDP. In such a context, ε-optimal strategies can be chosen among deterministic strategies (see for instance the explanation in [9, Thm. 1]).

Definition 14 (Positional, Deterministic strategies). *A Player* A *strategy* s_A *is* positional *if, for all states* $q \in Q$ *and paths* $\rho \in Q^+$ *we have* $\mathsf{s}_\mathsf{A}(\rho \cdot q) = \mathsf{s}_\mathsf{A}(q)$.

A Player B *strategy* s_B *is* deterministic *if, for all finite sequences* $\rho \cdot q \in Q^+$, *there exists* $b \in B_q$ *such that* $\mathsf{s}_\mathsf{B}(\rho \cdot q)(b) = 1$.

5 Necessary and sufficient condition for subgame optimality

In this section, we present a necessary and sufficient pair of conditions for a Player A strategy to be subgame optimal, formally stated in Theorem 1. The arguments given here are somewhat similar to the ones given in Section 4 of [4], which deals with the same question restricted to positional strategies.

The first condition is local: it specifies how a strategy behaves in the games in normal form at each local interaction of the game. As mentioned in Proposition 1, at each state q, the value of the game in normal form $\mathcal{F}_q^{\mathsf{nf}}$ is equal to the value of the state q (given by the valuation $\chi_\mathcal{G} \in [0,1]^Q$). This suggests that, for all finite sequences of states $\rho \in Q^+$ ending at that state q, the GF-strategy $\mathsf{s}_\mathsf{A}(\rho)$ needs to be optimal in the game in normal form $\mathcal{F}_q^{\mathsf{nf}}$ for the residual strategy $\mathsf{s}_\mathsf{A}^\rho$ to be optimal from q. Strategies with such a property are called locally optimal. This is a necessary condition for subgame optimality. (However, it is neither a necessary nor a sufficient condition for optimality, as argued in Section 6).

Definition 15 (Locally optimal strategies). *Consider a PI concurrent game* $\mathcal{G}$. *A Player* A *strategy* s_A *is locally optimal if, for all* $\rho = \rho' \cdot q \in Q^+$, *the* GF*-strategy* $\mathsf{s}_\mathsf{A}(\rho)$ *is optimal in the game in normal form* $\mathcal{F}_q^{\mathsf{nf}}$. *That is – recalling that* $\nu_\mathcal{G} \in [0,1]^\mathsf{D}$ *lifts the valuation* $\chi_\mathcal{G} \in [0,1]^Q$ *to the Nature states – for all* $b \in B_q$*:* $\chi_\mathcal{G}(q) \leq \sum_{a \in A} \mathsf{s}_\mathsf{A}(\rho)(a) \cdot \nu_\mathcal{G} \circ \delta(q,a,b) = \mathsf{out}_{\mathcal{F}_q^{\mathsf{nf}}}(\mathsf{s}_\mathsf{A}(\rho), b)$

Lemma 1. *In a PI concurrent game, subgame optimal strategies are locally optimal.*

Note that this was already shown for positional strategies in [4].

Local optimality does not ensure subgame optimality in general. However, it does ensure that, for all Player B deterministic strategies, the game almost-surely eventually settles in a value area, i.e. in some Q_u for some $u \in V_\mathcal{G}$.

Lemma 2. *Consider a PI concurrent game* $\mathcal{G}$ *and a Player* A *locally optimal strategy* s_A. *For all Player* B *deterministic strategies, almost surely the states seen infinitely often have the same value. That is:* $\mathbb{P}^{\mathsf{s}_\mathsf{A},\mathsf{s}_\mathsf{B}}[\bigcup_{u \in V_\mathcal{G}} Q^* \cdot (Q_u)^\omega] = 1$.

Proof (Sketch). First, if a state of value 1 is reached (i.e. a state in Q_1), then all states that can be seen with positive probability have value 1 (i.e. are in Q_1), since the strategy s_A is locally optimal. Let now $u \in V_\mathcal{G}$ be the highest value in $V_\mathcal{G}$ that is not 1 and consider the set of infinite paths such that the set Q_u is seen infinitely often but the game does not settle in it, i.e. the set $(Q^* \cdot (Q \setminus Q_u))^\omega \cap (Q^* \cdot Q_u)^\omega \subseteq Q^\omega$. Since the strategy s_A is locally optimal (and

since $V_{\mathcal{G}}$ is finite), one can show that there is a positive probability $p > 0$ such that, the conditional probability of reaching Q_1 knowing that Q_u is left is at least p. Hence, if Q_u is left infinitely often, almost-surely the set Q_1 is seen (and never left). It follows that the probability of the event $(Q^* \cdot (Q \setminus Q_u))^\omega \cap (Q^* \cdot Q_u)^\omega$ is 0. This implies that, almost-surely, if the set Q_u is seen infinitely often, then at some point it is never left. The same arguments can then be used with the highest value in $V_{\mathcal{G}}$ that is less than u, etc. Overall, we obtain that, for all $u \in V_{\mathcal{G}}$, if a set Q_u is seen infinitely often, it is eventually never left almost-surely.

Local optimality ensures that, at each step, the expected values of the states reached does not worsen (and may even improve if Player B does not play optimally). By propagating this property, we obtain that, given a Player A locally optimal strategy and a Player B deterministic strategy, the convex combination of the values u in $V_{\mathcal{G}}$ weighted by the probability of settling in the value area Q_u, from a state q is at least equal to its value $\chi_{\mathcal{G}}(q)$. This is stated in Lemma 3 below.

Lemma 3. *For a PI concurrent game $\mathcal{G}$, a Player* A *locally optimal strategy* s_A*, a Player* B *deterministic strategy* s_B *and a state $q \in Q$: $\chi_{\mathcal{G}}(q) \leq \sum_{u \in V_{\mathcal{G}}} u \cdot \mathbb{P}_q^{\mathsf{s}_\mathsf{A}, \mathsf{s}_\mathsf{B}}[Q^* \cdot (Q_u)^\omega]$.*

Note that if Player B plays subgame optimally, then this inequality is an equality.

Proof (Sketch). First, let us denote $\mathbb{P}_q^{\mathsf{s}_\mathsf{A}, \mathsf{s}_\mathsf{B}}$ by $\mathbb{P}$. It can be shown by induction that, for all $i \in \mathbb{N}^*$, we have the property $\mathcal{P}(i) : \chi_{\mathcal{G}}(q) \leq \sum_{\pi \cdot q' \in q \cdot Q^i} \chi_{\mathcal{G}}(q') \cdot \mathbb{P}(\pi \cdot q') = \sum_{u \in V_{\mathcal{G}} \setminus \{0\}} u \cdot \mathbb{P}[q \cdot Q^{i-1} \cdot Q_u]$. Furthermore, since by Lemma 2, the game almost-surely settles in a value area, it can be shown that for n large enough, the probability of being in Q_u after n steps (i.e. $\mathbb{P}[q \cdot Q^{n-1} \cdot Q_u]$) is arbitrarily close to the probability of eventually settling in Q_u (i.e. $\mathbb{P}[Q^* \cdot (Q_u)^\omega]$). We can then apply $\mathcal{P}(n)$ to obtain the desired inequality.

Recall that we are considering a pair of conditions to characterize that a strategy is subgame optimal. The first condition is local optimality. To summarize, we have seen that the fact that a strategy is locally optimal ensures that, from any state q, the expected values of the value areas where the game settles is at least $\chi_{\mathcal{G}}(q)$. However, local optimality does not ensure anything as to the probability of W given that the game settles in a specific value area. This is where the second condition comes into play. For the explanations regarding this condition, we will need Lemma 4 below: a consequence of Levy's 0-1 Law.

Lemma 4. *Let $\mathcal{M}$ be a countable Markov chain with a PI objective. If there is a $q \in Q$ such that $\chi_{\mathcal{M}}(q) < 1$, then $\inf_{q' \in Q} \chi_{\mathcal{M}}(q') = 0$.*

Consider now a Player A subgame optimal strategy s_A and a Player B deterministic strategy. Let us consider what happens if the game eventually settles in Q_u for some $u \in V_{\mathcal{G}} \setminus \{0\}$. Assume towards a contradiction that there is a finite path after which the probability of W given that the play eventually settles in Q_u is less than 1. Then, there is a continuation of this path ending in Q_u for which this probability of W is less than u. Indeed, it was shown that, for a PI objective,

in a countable Markov chain (which is what we obtain once strategies for both players are fixed), if there is a state with a value less than 1, then the infimum of the values in the Markov chain is 0 (this is what is stated in Lemma 4). Following our above towards-a-contradiction-assumption, there would be a finite path from which the Player A strategy s_A is not optimal. This is in contradiction with the fact that it is subgame optimal. Hence, a second necessary condition – in addition to the local optimality assumption – for subgame optimality is: from all finite paths, for all Player B deterministic strategies, for all positive values $u \in V_\mathcal{G} \setminus \{0\}$, the probability of W and eventually settling in Q_u is equal to the probability of eventually settling in Q_u. We obtain the theorem below.

Theorem 1. *Consider a concurrent game $\mathcal{G}$ with a PI objective W and a Player A strategy $\mathsf{s}_\mathsf{A} \in \mathsf{S}^\mathsf{A}_\mathcal{C}$. The strategy s_A is subgame optimal if and only if:*

- *it is locally optimal;*
- *for all $\rho \in Q^+$, for all Player B deterministic strategies s_B, for all values $u \in V_\mathcal{G} \setminus \{0\}$, we have $\mathbb{P}^{\mathsf{s}^\rho_\mathsf{A},\mathsf{s}^\rho_\mathsf{B}}_\rho[W \cap Q^* \cdot (Q_u)^\omega] = \mathbb{P}^{\mathsf{s}^\rho_\mathsf{A},\mathsf{s}^\rho_\mathsf{B}}_\rho[Q^* \cdot (Q_u)^\omega]$.*

Proof (Sketch). Lemma 1 states that local optimality is necessary and we have informally argued above why the second condition is also necessary for subgame optimality. As for the fact that they are sufficient conditions, this is a direct consequence of Lemmas 2 and 3 and the fact that deterministic strategies can achieve the same values as arbitrary strategies in MDPs (which we obtain once a Player A strategy is fixed), as cited in Subsection 4.2.

One may ask what happens in the special case where the strategy s_A considered is positional. As mentioned above, such a characterization was already presented in [4][1]. Overall, we obtain a similar result except that the second condition is replaced by what happens in the game restricted to the End Components in the Markov Decision Process induced by the positional strategy s_A.

6 From subgame almost-surely winning to subgame optimality

In [14, Thm. 4.5], the authors have proved a transfer result in PI turn-based games: the amount of memory sufficient to play optimally in every state of value 1 of every game is also sufficient to play optimally in every game. This result does not hold on concurrent games as is. First, although there are always optimal strategies in PI turn-based games (as proved in the same paper [14, Thm. 4.3]), there are PI concurrent games without optimal strategies. Second, infinite memory may be required to play optimally in co-Büchi concurrent games whereas almost-surely winning strategies can be found among positional strategies in a turn-based setting. This can be seen in the game of Figure 1 with $\mathsf{col}(q_0) = 0$ and $\mathsf{col}(q_1) = \mathsf{col}(q'_1) = 1$. The green values in the local interaction at state q_0 are the

[1] The proof was only presented for a specific class of objectives.

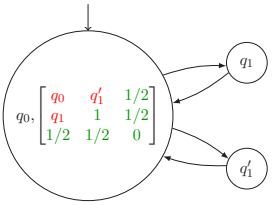


Fig. 1. A co-Büchi game.

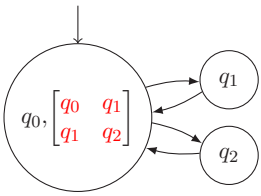


Fig. 2. A parity game.

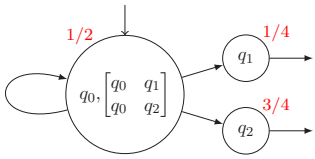


Fig. 3. A concurrent game with $A_{q_0} = \{a_1, a_2\}$.

values of the game if they are reached (the game ends immediately). If a green value is not reached, the objective of Player A is to see only finitely often states q_1 and q_1'. It has already been argued in [4] that the value of this game is $1/2$ and that there is an optimal strategy for Player A but it requires infinite memory. To play optimally, Player A must play the top row with probability $1 - \varepsilon_k$ and the middle row with probability ε_k for $\varepsilon_k > 0$ that goes (fast) to 0 when k goes to ∞ (where k denotes the number of steps). The ε_k must be chosen so that, if Player B always plays the left column with probability 1, then the state q_1 is seen finitely often with probability 1. Furthermore, as soon as the state q_1' is visited, Player A switches to a positional strategy playing the bottom row with probability ε_k' small enough (where k denotes the number of steps before the state q_1' was seen) and the two top rows with probability $(1 - \varepsilon_k')/2$.

Hence, the transfer of memory from almost-surely winning to optimal does not hold in concurrent games even if it is assumed that optimal strategies exist. However, one can note that although the strategy described above is optimal, it is not subgame optimal. Indeed, when the strategy switches, the value of the residual strategy is $1/2 - \varepsilon_k' < 1/2$. In fact, there is no subgame optimal strategy in that game. Actually, if we assume that, not only optimal but subgame optimal strategies exist, then the transfer of memory will hold.

The aim of this section is twofold: first, we identify a necessary and sufficient condition for the existence of subgame optimal strategies[2]. Second, we establish the above-mentioned memory transfer that relates the amount of memory to play subgame optimally and to be almost-surely winning. Before stating the main theorem of this section, let us first introduce the definition of positionally subgame almost-surely winnable objective, i.e. objectives for which subgame almost-surely winning strategies can be found among positional strategies.

Definition 16 (Positionally subgame almost-surely winnable objective). *Consider a PI objective $W \subseteq \mathsf{K}^\omega$. It is said to be a positionally subgame almost-surely winnable objective (PSAW for short) if the following holds: in all concur-*

[2] Note that this is different from what we did in the previous section: there, we established a necessary and sufficient condition for a specific strategy to be subgame optimal. Here, given a game, we consider necessary and sufficient conditions on the game for the existence of a subgame optimal strategy.

rent games $\mathcal{G} = \langle \mathcal{C}, W \rangle$ where there is a subgame almost-surely winning strategy, there is a positional one.

Theorem 2. *Consider a non-empty finite set of colors K and a PI objective $\emptyset \subsetneq W \subseteq \mathsf{K}^\omega$. Consider a concurrent game $\mathcal{G}$ with objective W. Then, the three following assertions are equivalent:*

a. there exists a subgame optimal strategy;
b. there exists an optimal strategy that is locally optimal;
c. there exists a positively-optimal strategy that is locally optimal.

Furthermore, if this holds and if the objective W is PSAW, then there exists a subgame optimal positional strategy.

First, note that the equivalence is stated in terms of existence of strategies, not on the strategies themselves. In particular, any subgame optimal strategy is both optimal and locally optimal, however, an optimal strategy that is locally optimal is not necessarily a subgame optimal strategy. Second, it is straightforward that point *a* implies point *b* (from Theorem 1) and that point *b* implies point *c* (by definition of positively-optimal strategies). In the remainder of this section, we explain informally the constructions leading to the proof of this theorem, i.e. to the proof that point *c* implies point *a*. The transfer of memory is a direct consequence of the way this theorem is proven. We fix a PI concurrent game $\mathcal{G} = \langle \mathcal{C}, W \rangle$ for the rest of the section.

The idea is as follows. As stated in Theorem 1, subgame optimal strategies are locally optimal and win the game almost-surely if the game settles in a value area Q_u for some positive $u \in V_{\mathcal{G}} \setminus \{0\}$. Our idea is therefore to consider subgame almost-surely winning strategies in the derived game $\mathcal{G}_u$: a "restriction" of the game $\mathcal{G}$ to Q_u (more details will be given later). We can then glue together these subgame almost-surely winning strategies – defined for all $u \in V_{\mathcal{G}} \setminus \{0\}$ – into a subgame optimal strategy. However, there are some issues:

1. the state values in the game $\mathcal{G}_u$ should be all equal to 1;
2. furthermore, there must exist a subgame almost-surely winning strategy in $\mathcal{G}_u$;
3. this subgame almost-surely winning strategy in $\mathcal{G}_u$ should be locally optimal when considered in the whole game $\mathcal{G}$.

Note that the method we use here is different from what the authors of [14] did to prove the transfer of memory in turn-based games.

Let us first deal with issue **3**. One can ensure that the almost-surely winning strategies in the game $\mathcal{G}_u$ are all locally optimal in $\mathcal{G}$ by properly defining the game $\mathcal{G}_u$. More specifically, this is done by enforcing that the only Player A possible strategies in $\mathcal{G}_u$ are locally optimal in the game $\mathcal{G}$. To do so, we construct the game $\mathcal{G}_u$ whose state space is Q_u (plus gadget states) but whose set of actions $A_{\mathcal{F}^{\mathsf{nf}}_q}$, at a state $q \in Q_u$, is such that the set of strategies $\mathcal{D}(A_{\mathcal{F}^{\mathsf{nf}}_q})$ corresponds exactly to the set of optimal strategies in the original game in normal form $\mathcal{F}^{\mathsf{nf}}_q$, while keeping the set of actions $A_{\mathcal{F}^{\mathsf{nf}}_q}$ for Player A finite. This is possible thanks

$$\begin{array}{c} \\ a_1 \\ a_2 \end{array}\!\!\begin{array}{c} \begin{array}{cc} \quad & \quad \end{array} \\ \begin{bmatrix} q_0 & q_1 \\ q_0 & q_2 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ a_1 \\ a_2 \end{array}\!\!\begin{array}{c} \begin{array}{cc} \quad & \quad \end{array} \\ \begin{bmatrix} \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & \frac{3}{4} \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ \frac{a_1+a_2}{2} \\ a_2 \end{array}\!\!\begin{array}{c} \begin{array}{cc} \quad & \quad \end{array} \\ \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{3}{4} \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ \frac{a_1+a_2}{2} \\ a_2 \end{array}\!\!\begin{array}{c} \begin{array}{cc} \quad & \quad \end{array} \\ \begin{bmatrix} q_0 & \frac{q_1+q_2}{2} \\ q_0 & q_2 \end{bmatrix} \end{array}$$

Fig. 4. The local interaction $\mathcal{F}_{q_0}$ at state q_0.

Fig. 5. The game in normal form $\mathcal{F}^{\mathsf{nf}}_{q_0}$ at the state q_0.

Fig. 6. The game $\mathcal{F}^{\mathsf{opt,nf}}_{q_0}$ with only optimal strategies.

Fig. 7. The game form $\mathcal{F}^{\mathsf{opt}}_{q_0}$ with only optimal strategies.

to Proposition 2 below: in every game in normal form $\mathcal{F}^{\mathsf{nf}}_q$ at state $q \in Q_u$, there exists a finite set $A_{\mathcal{F}^{\mathsf{nf}}_q}$ of optimal strategies such that the optimal strategies in $\mathcal{F}^{\mathsf{nf}}_q$ are exactly the convex combinations of strategies in $A_{\mathcal{F}^{\mathsf{nf}}_q}$. This is a well known result, argued for instance in [18].

Proposition 2. *Consider a game in normal form $\mathcal{F}^{\mathsf{nf}} = \langle A, B, [0,1], \delta \rangle$ with $|A| = n$ and $|B| = k$. There exists a set $A_{\mathcal{F}^{\mathsf{nf}}} \subseteq \mathsf{Opt}_{\mathsf{A}}(\mathcal{F}^{\mathsf{nf}})$ of optimal strategies such that $|A_{\mathcal{F}^{\mathsf{nf}}}| \leq n + k$ and $\mathcal{D}(A_{\mathcal{F}^{\mathsf{nf}}}) = \mathsf{Opt}_{\mathsf{A}}(\mathcal{F}^{\mathsf{nf}})$.*

Proof (Sketch). One can write a system of $n + k$ inequalities (with some additional equalities) whose set of solutions is exactly the set of optimal GF-strategies $\mathsf{Opt}_{\mathsf{A}}(\mathcal{F}^{\mathsf{nf}})$. The result then follows from standard system of inequalities arguments as the space of solutions is in fact a polytope with at most $n+k$ vertices. .

We illustrate this construction: a part of a concurrent game is depicted in Figure 3 and the change of the interaction of the players at state q_0 is depicted in Figures 4, 5, 6 and 7.

The game $\mathcal{G}_u$ has the same objective W as the game $\mathcal{G}$. Since we want all the states to have value 1 in $\mathcal{G}_u$ (recall issue **1**), we will build the game $\mathcal{G}_u$ such that any edge leading to a state not in Q_u in $\mathcal{G}$ now leads to a PI concurrent game $\mathcal{G}_W$ (with the same objective W) where all states have value 1. The game $\mathcal{G}_W$ is (for instance) a clique with all colors in K where Player A plays alone.

An illustration of this construction can be found in Figures 8 and 9. The blue dotted arrows are the ones that need to be redirected when the game is changed. With such a definition, we have made some progress w.r.t. the issue **1** cited previously (regarding the values being equal to 1): the values of all states of the game $\mathcal{G}_u$ are positive (for positive u).

Lemma 5. *Consider the game $\mathcal{G}_u$ for some positive $u \in V_{\mathcal{G}} \setminus \{0\}$ and assume that, in $\mathcal{G}$, there exists a positively-optimal strategy that is locally optimal. Then, for all states q in $\mathcal{G}_u$, the value of the state q in $\mathcal{G}_u$ is positive: $\chi_{\mathcal{G}_u}(q) > 0$.*

Proof (Sketch). Consider a state $q \in Q_u$ and a Player A locally optimal strategy s_{A} in $\mathcal{G}$ that is positively-optimal from q. Then, the strategy s_{A} (restricted to Q_u^+) can be seen as a strategy in $\mathcal{G}_u$ (it has to be defined in $\mathcal{G}_W$, but this can done straightforwardly). Note that this is only possible because the strategy s_{A} is locally optimal (due to the definition of $\mathcal{G}_u$). For a Player B strategy s_{B} in $\mathcal{G}_u$, consider what happens with strategies s_{A} and s_{B} in both games $\mathcal{G}_u$ and $\mathcal{G}$. Either

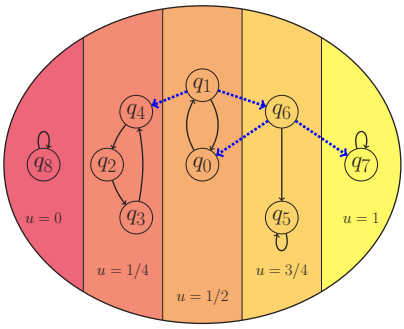


Fig. 8. The depiction of a PI concurrent game with its value areas.

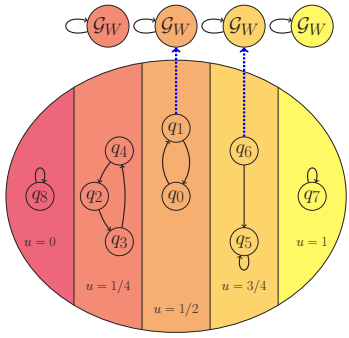


Fig. 9. The PI concurrent game after the modifications described above.

the game stays indefinitely in Q_u, and what happens in $\mathcal{G}_u$ and $\mathcal{G}$ is identical. Or it eventually leaves Q_u, leading to states of value 1 in $\mathcal{G}_u$. Hence, the value of the game $\mathcal{G}_u$ from q with strategies s_A and s_B is at least the value of the game $\mathcal{G}$ from q with the same strategies. Thus, the value of the state q is positive in $\mathcal{G}_u$.

As it turns out, Lemma 5 suffices to deal with both issues **1** and **2** at the same time. Indeed, as stated in Theorem 3 below, it is a general result that in a PI concurrent game, if all states have positive values, then all states have value 1 and there is a subgame almost-surely winning strategy.

Theorem 3. *Consider a PI concurrent game $\mathcal{G}$ and assume that all state values are greater than or equal to $c > 0$, i.e. for all $q \in Q$, $\chi_\mathcal{G}(q) \geq c$. Then, there is a subgame almost-surely winning strategy in $\mathcal{G}$.*

Remark 1. This theorem can be seen as a strengthening of Theorem 1 from [6]. Indeed, this Theorem 1 states that if all states have positive values, then they all have value 1 (this is then generalized to games with countably-many states). Theorem 3 is stronger since it ensures the existence of (subgame) almost-surely winning strategies. Although a detailed proof is provided in the complete version of this paper [5], note that this theorem was already stated and proven in [14] in the context of PI turn-based games. Nevertheless their arguments could have been used *verbatim* for concurrent games as well. In [5], we give a proof using the same construction (namely, reset strategies) but we argue differently why the construction proves the theorem.

We can now glue together pieces of strategies s^u_A defined in all games $\mathcal{G}_u$ into a single strategy $\mathsf{s}_\mathsf{A}[(\mathsf{s}^u_\mathsf{A})_{u \in V_\mathcal{G} \setminus \{0\}}]$. Informally, the glued strategy mimics the strategy on Q^+_u and switches strategy when a value area is left and another one is reached.

Definition 17 (Gluing strategies). *Consider a PI concurrent game $\mathcal{G}$ and for all values $u \in V_\mathcal{G} \setminus \{0\}$, a strategy s^u_A in the game $\mathcal{G}_u$. Then, we glue these*

strategies into the strategy $\mathsf{s}_{\mathsf{A}}[(\mathsf{s}^u_{\mathsf{A}})_{u\in V_{\mathcal{G}}\setminus\{0\}}] : Q^+ \to \mathcal{D}(A)$ *simply written* s_{A} *such that, for all* ρ *ending at state* $q \in Q$*:*

$$\mathsf{s}_{\mathsf{A}}(\rho) := \begin{cases} \mathsf{s}^u_{\mathsf{A}}(\pi) & \text{if } u = \chi_{\mathcal{G}}(q) > 0 \text{ for } \pi \text{ the longest suffix of } \rho \text{ in } Q^+_u \\ \text{is arbitrary} & \text{if } \chi_{\mathcal{G}}(q) = 0 \end{cases}$$

As stated in Lemma 6 below, the construction described in Definition 17 transfers almost-surely winning strategies in $\mathcal{G}_u$ into a subgame optimal strategy in $\mathcal{G}$.

Lemma 6. *For all* $u \in V_{\mathcal{G}}\setminus\{0\}$*, let* $\mathsf{s}^u_{\mathsf{A}}$ *be a subgame almost-surely winning strategy in* $\mathcal{G}_u$*. The glued strategy* $\mathsf{s}_{\mathsf{A}}[(\mathsf{s}^u_{\mathsf{A}})_{u\in V_{\mathcal{G}}\setminus\{0\}}]$*, denoted* s_{A}*, is subgame optimal in* $\mathcal{G}$*.*

Proof (Sketch). We apply Theorem 1. First, the strategy s_{A} is locally optimal in all Q_u for $u > 0$ by the strategy restriction done to define the game $\mathcal{G}_u$ (only optimal strategies are considered at each game in normal form $\mathcal{F}^{\mathsf{nf}}_q$ at states $q \in Q_u$). Furthermore, any strategy is optimal in a game in normal form of value 0 (which is the case of the game in normal forms of states in Q_0). Second, if the game eventually settles in a value area Q_u for some $u > 0$, from then on the strategy s_{A} mimics the strategy $\mathsf{s}^u_{\mathsf{A}}$, which is subgame almost-surely winning in $\mathcal{G}_u$. Hence, the probability of W given that the game eventually settles in Q_u is 1. This holds for all $u \in V_{\mathcal{G}} \setminus \{0\}$, so the second condition of Theorem 1 holds.

We now have all the ingredients to prove Theorem 2.

Proof (Of Theorem 2). We consider the PI concurrent game $\mathcal{G}$ and assume that there is a positively-optimal strategy that is locally optimal. Then, by Lemma 5, for all positive values $u \in V_{\mathcal{G}}\setminus\{0\}$, all states in $\mathcal{G}_u$ have positive values. It follows, by Theorem 3, that there exists a subgame almost-surely winning strategy in every game $\mathcal{G}_u$ for $u \in V_{\mathcal{G}} \setminus \{0\}$. We then obtain a subgame optimal strategy by gluing these strategies together, given by Lemma 6.

The second part of the theorem, dealing with transfer of positionality from subgame almost-surely winning to subgame optimal follows from the fact that if all strategies $\mathsf{s}^u_{\mathsf{A}}$ are positional for all $u \in V_{\mathcal{G}} \setminus \{0\}$, then so is the glued strategy $\mathsf{s}_{\mathsf{A}}[(\mathsf{s}^u_{\mathsf{A}})_{u\in V_{\mathcal{G}}\setminus\{0\}}]$.

We now apply the result of Theorem 2 to two specific classes of objectives: Büchi and co-Büchi objectives. Note that this result is already known for Büchi objectives, proven in [4].

Corollary 1. *Consider a concurrent game with a Büchi (resp. co-Büchi) objective and assume that there is a positively-optimal strategy that is locally optimal. Then there is a subgame optimal positional strategy.*

Note that it is also possible to prove a memory transfer from subgame almost-surely winning to subgame optimal for an arbitrary memory skeleton, instead of only positional strategies. This adds only a few minor difficulties.

Application to the turn-based setting. The aim of Section 6 was to extend an already existing result on turn-based games in the context of concurrent games. This required an adaptation of the assumptions. However, it is in fact possible to retrieve the original result on turn-based games from Theorem 2 in a fairly straightforward manner. It amounts to show that, in all finite turn-based games $\mathcal{G}$, for all values $u \in V_{\mathcal{G}} \setminus \{0\}$, there is a locally optimal strategy that is positively-optimal from all states in Q_u.

7 Finite-choice strategies

In this section, we introduce a new kind of strategies, namely finite-choice strategies. Let us first motivate why we consider such strategies. Consider again the co-Büchi game of Figure 1. Recall that the optimal strategy we described first plays the top row with increasing probability and the middle row with decreasing probability and then, once Player B plays the second column, switches to a positional strategy playing the bottom row with positive, yet small enough probability. Note that switching strategy is essential. Indeed, if Player A does not switch, Player B could at some point opt for the middle column and see indefinitely the state q_1' with very high probability. In fact, what happens in that case is rather counter-intuitive: once Player B switches, there is infinitely often a positive probability to reach the outcome of value 1. However, the probability to ever reaching this outcome can be arbitrarily small, if Player B waits long enough before playing the middle row. This happens because the probability ε_k to visit that outcome goes (fast) to 0 when k goes to ∞. In fact, such an optimal strategy has "infinite choice" in the sense that it may prescribe infinitely many different probability distribution.

In this section, we consider *finite-choice strategies*, i.e. strategies that can use only finitely many GF-strategies at each state.

Definition 18 (Finite-choice strategy). *Let $\mathcal{G}$ be a concurrent game. A Player A strategy s_A in $\mathcal{G}$ has* finite choice *if, for all $q \in Q$, the set $S_q^{\mathsf{s}_\mathsf{A}} := \{\mathsf{s}_\mathsf{A}(\rho \cdot q) \mid \rho \in Q^+\} \subseteq \mathcal{D}(A_q)$ is finite.*

Note that positional (even finite-memory) and deterministic strategies are examples of finite-choice strategies.

Interestingly, we can link finite-choice strategies with the existence of subgame optimal strategies. In general it does not hold that if there are optimal strategies, then there exists subgame optimal strategies (as exemplified in the game of Figure 1). However, in Theorem 4 below, we state that if we additionally assume that the optimal strategy considered has finite choice, then there is a subgame optimal strategy (that has also finite choice).

Theorem 4. *Consider a PI concurrent game $\mathcal{G}$. If there is a finite-choice optimal strategy, then there is a finite-choice subgame optimal strategy.*

Proof (Sketch). Consider such an optimal finite-choice strategy s_A. In particular, note that there is a constant $c > 0$ such that for all $\rho \cdot q \in Q^+$, for all $a \in A_q$ we

have: $\mathsf{s}_\mathsf{A}(\rho \cdot q)(q) > 0 \Rightarrow \mathsf{s}_\mathsf{A}(\rho \cdot q)(q) \geq c$. We build a subgame optimal strategy s'_A in the following way: for all $\rho = \rho' \cdot q \in Q^+$, if the residual strategy $\mathsf{s}^\rho_\mathsf{A}$ is optimal, then $\mathsf{s}'_\mathsf{A}(\rho) := \mathsf{s}_\mathsf{A}(\rho)$, otherwise $\mathsf{s}'_\mathsf{A}(\rho) := \mathsf{s}_\mathsf{A}(q)$ (i.e. we reset the strategy). Straightforwardly, the strategy s'_A has finite choice. We want to apply Theorem 1 to prove that it is subgame optimal. One can see that it is locally optimal (by the criterion chosen for resetting the strategy). Consider now some $\rho \in Q^+$ ending at state $q \in Q$ and another state $q' \in Q$. Assume that the residual strategy $\mathsf{s}^\rho_\mathsf{A}$ is optimal but that the residual strategy $\mathsf{s}^{\rho \cdot q'}_\mathsf{A}$ is not. Then, similarly to why local optimality is necessary for subgame optimality (see Proposition 1), one can show that any Player B action b leading to q' from ρ with positive probability is such that $\chi_\mathcal{G}(q) < \mathsf{out}_{\mathcal{F}^{\mathsf{nf}}_q}(\mathsf{s}_\mathsf{A}(\rho), b)$. Hence, there is positive probability from ρ, if Player B opts for the action b, to reach a state of value different from $u = \chi_\mathcal{G}(q)$. And if this happens infinitely often, a state of value different from u will be reached almost-surely[3]. In other words, if a value area is never left, almost-surely, the strategy s'_A only resets finitely often.

Consider now some $\rho \in Q^+$, a Player B deterministic strategy s_B and a value $u \in V_\mathcal{G} \setminus \{0\}$. From what we argued above, the probability of the event $Q^* \cdot (Q_u)^\omega$ (resp. $W \cap Q^* \cdot (Q_u)^\omega$) is the same if we intersect it with the fact that the strategy s'_A only resets finitely often. Furthermore, if the strategy does not reset anymore from some point on, and all states have the same value $u > 0$, then it follows that the probability of W is 1 (since W is PI). We can then conclude by applying Theorem 1.

Finite-choice strategies are interesting for another reason. In the previous section, we applied the memory transfer from Theorem 2 to the Büchi and co-Büchi objectives. We did not apply it to other objectives – in particular to the parity objective. Indeed, in general, contrary to the case of turn-based games, infinite-memory is necessary to be almost-surely winning in parity games. This happens in Figure 2 (already described in [12]) where the objective of Player A is to see q_1 infinitely often, while seeing q_2 only finitely often. Let us describe a Player A subgame almost-surely winning strategy. The top row is played with probability $1 - \varepsilon_k$ and the bottom row is played with probability $\varepsilon_k > 0$ with ε_k going to 0 when k goes to ∞ (the (ε_k) used in the game in Figure 1 works here as well) where k denotes the number of times the state q_0 is seen. Such a strategy is subgame almost-surely winning and does not have finite choice. In fact, it can be shown that all Player A finite-choice strategies have value 0 in that game.

Interestingly, the transfer of memory of Theorem 2 is adapted in Theorem 5 with the memory that is sufficient in turn-based games – for those PI objectives that have a "neutral color"– if we additionally assume that the subgame optimal strategy considered has finite choice. First, let us define what is meant by "neutral color", then we define the turn-based version of PSAW.

[3] This holds because the strategy s_A has finite choice: the probability to see a state of different value is bounded below by the product of c and the smallest positive probability among all Nature states.

Definition 19 (Objective with a neutral color). *Consider a set of colors* K *and a PI objective* $W \subseteq \mathsf{K}^\omega$*. It has a* neutral color *if there is some (neutral) color* $k \in \mathsf{K}$ *such that, for all* $\rho = \rho_0 \cdot \rho_1 \cdots \in \mathsf{K}^\omega$*, we have* $\rho \in W \Leftrightarrow \rho_0 \cdot k \cdot \rho_1 \cdot k \cdots \in W$*.*

Definition 20 (PASW objective in turn-based games). *Consider a PI objective* $W \subseteq \mathsf{K}^\omega$*. It is* positionally subgame almost-surely winnable in turn-based games *(PSAWT for short) if in all turn-based games* $\mathcal{G} = \langle \mathcal{C}, W \rangle$ *where there is a subgame almost-surely winning strategy, there is a positional one.*

Theorem 5. *Consider a PSAWT PI objective* $W \subseteq \mathsf{K}^\omega$ *with a neutral color and a concurrent game* $\mathcal{G}$ *with objective* W*. Assume there is a subgame optimal strategy that has finite choice. Then, there is a positional one.*

Proof (Sketch). A finite-choice strategy s_A plays only among a finite number of GF-strategies at each state. The idea is therefore to modify the game $\mathcal{G}_u$ of the previous subsection into a game $\mathcal{G}'_u$ by transforming it into a (finite) turn-based game. At each state, Player A chooses first her GF-strategy. She can choose among only a finite number of them: she has at her disposal, at a state q, only optimal GF-strategies in $S_q^{\mathsf{s}_\mathsf{A}}$ (recall Definition 18). We consider the objective W in that new arena where Player B states are colored with a neutral color. The existence, in $\mathcal{G}$, of a subgame optimal strategy that has finite choice ensures that all states in $\mathcal{G}'_u$ have positive values. We can then conclude as for Theorem 2: a subgame optimal strategy can be obtained by gluing together subgame almost-surely winning strategies in the (turn-based) games $\mathcal{G}'_u$ (that can be chosen positional by assumption).

As an application, one can realize that the parity, mean-payoff and generalized Büchi objectives have a neutral color and are PSAWT ([11,16,7]). Hence, for these objectives, if there exists an optimal strategy that has finite choice, then there is one that is positional.

Corollary 2. *Consider a concurrent game* $\mathcal{G}$ *with a parity (resp. mean-payoff, resp. generalized Büchi) objective. Assume that there is an optimal strategy that has finite choice in* $\mathcal{G}$*. Then, there is a positional one.*

References

1. Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. *Handbook of Model Checking*, chapter Graph games and reactive synthesis, pages 921–962. Springer, 2018.
2. Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux. From local to global determinacy in concurrent graph games. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

3. Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux. Optimal strategies in concurrent reachability games. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 7:1–7:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
4. Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux. Playing (almost-)optimally in concurrent büchi and co-büchi games. *CoRR*, abs/2203.06966, 2022.
5. Benjamin Bordais, Patricia Bouyer, and Stéphane Le Roux. Sub-game optimal strategies in concurrent games with prefix-independent objectives. *CoRR*, abs/2301.10697, 2023.
6. Krishnendu Chatterjee. Concurrent games with tail objectives. *Theor. Comput. Sci.*, 388(1-3):181–198, 2007.
7. Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Trading memory for randomness. In *1st International Conference on Quantitative Evaluation of Systems (QEST 2004), 27-30 September 2004, Enschede, The Netherlands*, pages 206–217. IEEE Computer Society, 2004.
8. Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. The complexity of quantitative concurrent parity games. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 678–687. ACM Press, 2006.
9. Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Thomas A. Henzinger. Randomness for free. *Inf. Comput.*, 245:3–16, 2015.
10. Krishnendu Chatterjee and Rasmus Ibsen-Jensen. Qualitative analysis of concurrent mean-payoff games. *Inf. Comput.*, 242:2–24, 2015.
11. Krishnendu Chatterjee, Marcin Jurdzinski, and Thomas A. Henzinger. Quantitative stochastic parity games. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 121–130. SIAM, 2004.
12. Luca de Alfaro and Thomas A. Henzinger. Concurrent omega-regular games. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 141–154. IEEE Computer Society, 2000.
13. Hugh Everett. Recursive games. *Annals of Mathematics Studies – Contributions to the Theory of Games*, 3:67–78, 1957.
14. Hugo Gimbert and Florian Horn. Solving simple stochastic tail games. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 847–862. SIAM, 2010.
15. Marta Kwiatkowska, Gethin Norman, Dave Parker, and Gabriel Santos. Automatic verification of concurrent stochastic systems. *Formal Methods in System Design*, 58:188–250, 2021.
16. Thomas M Liggett and Steven A Lippman. Stochastic games with perfect information and time average payoff. *Siam Review*, 11(4):604–607, 1969.
17. Donald A. Martin. The determinacy of blackwell games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998.
18. Lloyd S Shapley and RN Snow. Basic solutions of discrete games. *Contributions to the Theory of Games*, 1(24):27–27, 1950.
19. Wolfgang Thomas. Infinite games and verification. In *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 58–64. Springer, 2002. Invited Tutorial.
20. John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton Univ. Press, Princeton, 1944.

Author Index

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 561–562, 2023.
https://doi.org/10.1007/978-3-031-30829-1