VERIFAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems

Tommaso Dreossi(✉), Daniel J. Fremont(✉), Shromona Ghosh(✉), Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia(✉)

University of California, Berkeley, USA
{tommasodreossi,dfremont,shromona.ghosh}@berkeley.edu, sseshia@eecs.berkeley.edu

Abstract. We present VERIFAI, a software toolkit for the formal design and analysis of systems that include artificial intelligence (AI) and machine learning (ML) components. VERIFAI particularly addresses challenges with applying formal methods to ML components such as perception systems based on deep neural networks, as well as systems containing them, and to model and analyze system behavior in the presence of environment uncertainty. We describe the initial version of VERIFAI, which centers on simulation-based verification and synthesis, guided by formal models and specifications. We give examples of several use cases, including temporal-logic falsification, model-based systematic fuzz testing, parameter synthesis, counterexample analysis, and data set augmentation.

Keywords: Formal methods · Falsification · Simulation · Cyber-physical systems · Machine learning · Artificial intelligence · Autonomous vehicles

1 Introduction

The increasing use of artificial intelligence (AI) and machine learning (ML) in systems, including safety-critical systems, has brought with it a pressing need for formal methods and tools for their design and verification. However, AI/ML-based systems, such as autonomous vehicles, have certain characteristics that make the application of formal methods very challenging. We mention three key challenges here; see Seshia et al. [23] for an in-depth discussion. First, several uses of AI/ML are for *perception*, the use of computational systems to mimic human perceptual tasks such as object recognition and classification, conversing in natural language, etc. For such perception components,

This work was supported in part by NSF grants 1545126 (VeHICaL), 1646208, 1739816, and 1837132, the DARPA BRASS program under agreement number FA8750-16-C0043, the DARPA Assured Autonomy program, the iCyPhy center, and Berkeley Deep Drive. NVIDIA Corporation donated the Titan Xp GPU used for this research.

T. Dreossi, D. J. Fremont, S. Ghosh—These authors contributed equally to the paper.

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 432–442, 2019.
https://doi.org/10.1007/978-3-030-25540-4_25

writing a formal specification is extremely difficult, if not impossible. Additionally, the signals processed by such components can be very high-dimensional, such as streams of images or LiDAR data. Second, *machine learning* being a dominant paradigm in AI, formal tools must be compatible with the data-driven design flow for ML and also be able to handle the complex, high-dimensional structures in ML components such as deep neural networks. Third, the *environments* in which AI/ML-based systems operate can be very complex, with considerable uncertainty even about how many (which) agents are in the environment (both human and robotic), let alone about their intentions and behaviors. As an example, consider the difficulty in modeling urban traffic environments in which an autonomous car must operate. Indeed, AI/ML is often introduced into these systems precisely to deal with such complexity and uncertainty! From a formal methods perspective, this makes it very hard to create realistic environment models with respect to which one can perform verification or synthesis.

In this paper, we introduce the VERIFAI toolkit, our initial attempt to address the three core challenges—perception, learning, and environments—that are outlined above. VERIFAI takes the following approach:

- *Perception:* A perception component maps a concrete feature space (e.g. pixels) to an output such as a classification, prediction, or state estimate. To deal with the lack of specification for perception components, VERIFAI analyzes them in the context of a closed-loop system using a system-level specification. Moreover, to scale to complex high-dimensional feature spaces, VERIFAI operates on an *abstract feature space* (or *semantic feature space*) [10] that describes semantic aspects of the environment being perceived, not the raw features such as pixels.
- *Learning:* VERIFAI aims to not only analyze the behavior of ML components but also use formal methods for their (re-)design. To this end, it provides features to (i) design the data set for training and testing [9], (ii) analyze counterexamples to gain insight into mistakes by the ML model, as well as (iii) synthesize parameters, including hyper-parameters for training algorithms and ML model parameters.
- *Environment Modeling:* Since it can be difficult, if not impossible, to exhaustively model the environments of AI-based systems, VERIFAI aims to provide ways to capture a designer's assumptions about the environment, including distribution assumptions made by ML components, and to describe the abstract feature space in an intuitive, declarative manner. To this end, VERIFAI provides users with SCENIC [12,13], a probabilistic programming language for modeling environments. SCENIC, combined with a renderer or simulator for generating sensor data, can produce semantically-consistent input for perception components.

VERIFAI is currently focused on AI-based cyber-physical systems (CPS), although its basic ideas can also be applied to other AI-based systems. As a pragmatic choice, we focus on simulation-based verification, where the simulator is treated as a black-box, so as to be broadly applicable to the range of simulators used in industry.[1] The input to

[1] Our work is complementary to the work on industrial-grade simulators for AI/ML-based CPS. In particular, VERIFAI enhances such simulators by providing formal methods for modeling (via the SCENIC language), analysis (via temporal logic falsification), and parameter synthesis (via property-directed hyper/model-parameter synthesis).

VERIFAI is a "closed-loop" CPS model, comprising a composition of the AI-based CPS system under verification with an environment model, and a property on the closed-loop model. The AI-based CPS typically comprises a perception component (not necessarily based on ML), a planner/controller, and the plant (i.e., the system under control). Given these, VERIFAI offers the following use cases: (1) temporal-logic falsification; (2) model-based fuzz testing; (3) counterexample-guided data augmentation; (4) counterexample (error table) analysis; (5) hyper-parameter synthesis, and (6) model parameter synthesis. The novelty of VERIFAI is that it is the first tool to offer this suite of use cases in an integrated fashion, unified by a common representation of an abstract feature space, with an accompanying modeling language and search algorithms over this feature space, all provided in a modular implementation. The algorithms and formalisms in VERIFAI are presented in papers published by the authors in other venues (e.g., [7–10,12,15,22]). The problem of temporal-logic falsification or simulation-based verification of CPS models is well studied and several tools exist (e.g. [3,11]); our work was the first to extend these techniques to CPS models with ML components [7,8]. Work on verification of ML components, especially neural networks (e.g., [14,26]), is complementary to the system-level analysis performed by VERIFAI. Fuzz testing based on formal models is common in software engineering (e.g. [16]) but our work is unique in the CPS context. Similarly, property-directed parameter synthesis has also been studied in the formal methods/CPS community, but our work is the first to apply these ideas to the synthesis of hyper-parameters for ML training and ML model parameters. Finally, to our knowledge, our work on augmenting training/test data sets [9], implemented in VERIFAI, is the first use of formal techniques for this purpose. In Sect. 2, we describe how the tool is structured so as to provide the above features. Sect. 3 illustrates the use cases via examples from the domain of autonomous driving.

2 VERIFAI Structure and Operation

VERIFAI is currently focused on simulation-based analysis and design of AI components for perception or control, potentially those using ML, in the context of a closed-loop cyber-physical system. Figure 1 depicts the structure and operation of the toolkit.

Inputs and Outputs: Using VERIFAI requires setting up a simulator for the domain of interest. As we explain in Sect. 3, we have experimented with multiple robotics simulators and provide an easy interface to connect a new simulator. The user then constructs the inputs to VERIFAI, including (i) a simulatable model of the system, including code for one or more controllers and perception components, and a dynamical model of the system being controlled; (ii) a probabilistic model of the environment, specifying constraints on the workspace, the locations of agents and objects, and the dynamical behavior of agents, and (iii) a property over the composition of the system and its environment. VERIFAI is implemented in Python for interoperability with ML/AI libraries and simulators across platforms. The code for the controller and perception component can be arbitrary executable code, invoked by the simulator. The environment model typically comprises a definition in the simulator of the different types of agents, plus a description of their initial conditions and other parameters using the SCENIC probabilistic programming language [12]. Finally, the property to be checked can be expressed

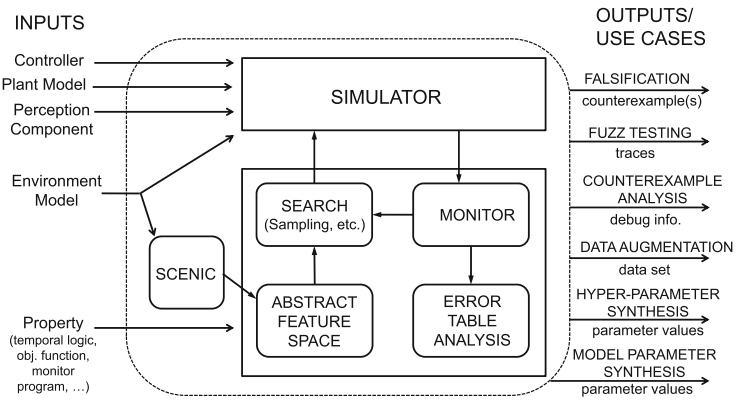


Fig. 1. Structure and operation of VERIFAI.

using Metric Temporal Logic (MTL) [2,24], objective functions, or arbitrary code monitoring the property. The output of VERIFAI depends on the feature being invoked. For falsification, VERIFAI returns one or more *counterexamples*, simulation traces violating the property [7]. For fuzz testing, VERIFAI produces traces sampled from the distribution of behaviors induced by the probabilistic environment model [12]. Error table analysis involves collecting counterexamples generated by the falsifier into a table, on which we perform analysis to identify features that are correlated with property failures. Data augmentation uses falsification and error table analysis to generate additional data for training and testing an ML component [9]. Finally, the property-driven synthesis of model parameters or hyper-parameters generates as output a parameter evaluation that satisfies the specified property.

Tool Structure: VERIFAI is composed of four main modules, as described below:

- *Abstract Feature Space and* SCENIC *Modeling Language:* The abstract feature space is a compact representation of the possible configurations of the simulation. Abstract features can represent parameters of the environment, controllers, or of ML components. For example, when analyzing a visual perception system for an autonomous car, an abstract feature space could consist of the initial poses and types of all vehicles on the road. Note that this abstract space, compared to the concrete feature space of pixels used as input to the controller, is better suited to the analysis of the overall closed-loop system (e.g. finding conditions under which the car might crash).

  VERIFAI provides two ways to construct abstract feature spaces. They can be constructed hierarchically, combining basic domains such as hyperboxes and finite sets into structures and arrays. For example, we could define a space for a car as a structure combining a 2D box for position with a 1D box for heading, and then create an array of these to get a space for several cars. Alternatively, VERIFAI allows a feature space to be defined using a program in the SCENIC language [12]. SCENIC provides convenient syntax for describing geometric configurations and agent parameters, and, as a probabilistic programming language, allows placing a distribution over the feature space which can be conditioned by declarative constraints.

- *Searching the Feature Space:* Once the abstract feature space is defined, the next step is to search that space to find simulations that violate the property or produce other interesting behaviors. Currently, VERIFAI uses a suite of sampling methods (both active and passive) for this purpose, but in the future we expect to also integrate directed or exhaustive search methods including those from the adversarial machine learning literature (e.g., see [10]). Passive samplers, which do not use any feedback from the simulation, include uniform random sampling, simulated annealing, and Halton sequences [18] (quasi-random deterministic sequences with low-discrepancy guarantees we found effective for falsification [7]). Distributions defined using SCENIC are also passive in this sense. Active samplers, whose selection of samples is informed by feedback from previous simulations, include cross-entropy sampling and Bayesian optimization. The former selects samples and updates the prior distribution by minimizing cross-entropy; the latter updates the prior from the posterior over a user-provided objective function, e.g. the satisfaction level of a specification or the loss of an analyzed model.
- *Property Monitor:* Trajectories generated by the simulator are evaluated by the monitor, which produces a score for a given property or objective function. VERIFAI supports monitoring MTL properties using the `py-metric-temporal-logic` [24] package, including both the Boolean and quantitative semantics of MTL. As mentioned above, the user can also specify a custom monitor as a Python function. The result of the monitor can be used to output falsifying traces and also as feedback to the search procedure to direct the sampling (search) towards falsifying scenarios.
- *Error Table Analysis:* Counterexamples are stored in a data structure called the error table, whose rows are counterexamples and columns are abstract features. The error table can be used offline to debug (explain) the generated counterexamples or online to drive the sampler towards particular areas of the abstract feature space. VERIFAI provides different techniques for error table analysis depending on the end use (e.g., counter-example analysis or data set augmentation), including principal component analysis (PCA) for ordered feature domains and subsets of the most recurrent values for unordered domains (see [9] for further details).

The communication between VERIFAI and the simulator is implemented in a client-server fashion using IPv4 sockets, where VERIFAI sends configurations to the simulator which then returns trajectories (traces). This architecture allows easy interfacing to a simulator and even with multiple simulators at the same time.

3 Features and Case Studies

This section illustrates the main features of VERIFAI through case studies demonstrating its various use cases and simulator interfaces. Specifically, we demonstrate model falsification and fuzz testing of an autonomous vehicle (AV) controller, data augmentation and error table analysis for a convolutional neural network, and model and hyperparameter tuning for a reinforcement learning-based controller.

3.1 Falsification and Fuzz Testing

VERIFAI offers a convenient way to debug systems through systematic testing. Given a model and a specification, the tool can use active sampling to automatically search for inputs driving the model towards a violation of the specification. VERIFAI can also perform model-based fuzz testing, exploring random variations of a scenario guided by formal constraints. To demonstrate falsification and fuzz testing, we consider two scenarios involving AVs simulated with the robotics simulator Webots [25]. For the experiments reported here, we used Webots 2018 which is commercial software.

In the first example, we falsify the controller of an AV which is responsible for safely maneuvering around a disabled car and traffic cones which are blocking the road. We implemented a hybrid controller which relies on perception modules for state estimation. Initially, the car follows its lane using standard computer vision (non-ML) techniques for line detection [20]. At the same time, a neural network (based on squeezeDet [27]) estimates the distance to the cones. When the distance drops below 15 m, the car performs a lane change, afterward switching back to lane-following.

The correctness of the AV is characterized by an MTL formula requiring the vehicle to maintain a minimum distance from the traffic cones and avoid overshoot while changing lanes. The task of the falsifier is to find small perturbations of the initial scene (generated by SCENIC) which cause the vehicle to violate this specification. We allowed perturbations of the initial positions and orientations of all objects, the color of the disabled car, and the cruising speed and reaction time of the ego car.

Our experiments showed that active samplers driven by the robustness of the MTL specification can efficiently discover scenes that confuse the controller and yield faulty behavior. Figure 2 shows an example, where the neural network detected the orange car instead of the traffic cones, causing the lane change to be initiated too early. As a result, the controller performed only an incomplete lane change, leading to a crash.

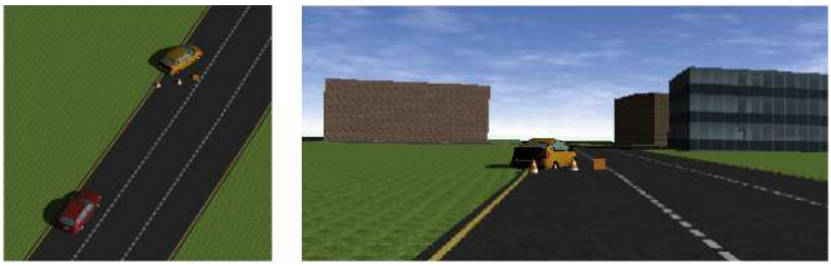


Fig. 2. A falsifying scene automatically discovered by VERIFAI. The neural network misclassifies the traffic cones because of the orange vehicle in the background, leading to a crash. Left: bird's-eye view. Right: dash-cam view, as processed by the neural network.

In our second experiment, we used VERIFAI to simulate variations on an actual accident involving an AV [5]. The AV, proceeding straight through an intersection, was hit by a human turning left. Neither car was able to see the other because of two lanes of stopped traffic. Figure 3 shows a (simplified) SCENIC program we wrote to reproduce

```
# Car going straight
ego = Car on egoLane.median

# Car turning left
Car on leftTurnLane.median

# A car blocking the Ego's view
spot = OrientedPoint on blockLane.median
laneNoise = (-0.5, 0.5)
Car at spot offset by laneNoise @ 0

# Another car 5-8 m behind that
Car at spot2 offset by laneNoise @ (-5, -8)
```

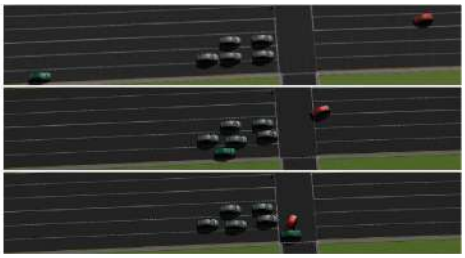


Fig. 3. Left: Partial SCENIC program for the crash scenario. `Car` is an object class defined in the Webots world model (not shown), `on` is a SCENIC *specifier* positioning the object uniformly at random in the given region (e.g. the median line of a lane), `(-0.5, 0.5)` indicates a uniform distribution over that interval, and `X @ Y` creates a vector with the given coordinates (see [12] for a complete description of SCENIC syntax). Right: (1) initial scene sampled from the program; (2) the red car begins its turn, unable to see the green car; (3) the resulting collision. (Color figure online)

the accident, allowing variation in the initial positions of the cars. We then ran simulations from random initial conditions sampled from the program, with the turning car using a controller trying to follow the ideal left-turn trajectory computed from OpenStreetMap data using the Intelligent Intersections Toolbox [17]. The car going straight used a controller which either maintained a constant velocity or began emergency breaking in response to a message from a simulated "smart intersection" warning about the turning car. By sampling variations on the initial conditions, we could determine how much advance notice is necessary for such a system to robustly avoid an accident.

3.2 Data Augmentation and Error Table Analysis

Data augmentation is the process of supplementing training sets with the goal of improving the performance of ML models. Typically, datasets are augmented with transformed versions of preexisting training examples. In [9], we showed that augmentation with counterexamples is also an effective method for model improvement.

Fig. 4. This image generated by our renderer was misclassified by the NN. The network reported detecting only one car when there were two.

VERIFAI implements a counterexample-guided augmentation scheme, where a falsifier (see Sect. 3.1) generates misclassified data points that are then used to augment the original training set. The user can choose among different sampling methods, with passive samplers suited to generating diverse sets of data points while active samplers can efficiently generate similar counterexamples. In addition to the counterexamples themselves, VERIFAI also returns an error table aggregating information on the misclassifications that can be used to drive the retraining process. Figure 4 shows the rendering of a misclassified sample generated by our falsifier.

For our experiments, we implemented a renderer that generates images of road scenarios and tested the quality of our augmentation scheme on the squeezeDet convolutional neural network [27], trained for classification. We adopted three techniques to select augmentation images: (1) randomly sampling from the error table, (2) selecting the top k-closest (similar) samples from the error table, and (3) using PCA analysis to generate new samples. For details on the renderer and the results of counterexample-driven augmentation, see [9]. We show that incorporating the generated counterexamples during re-training improves the accuracy of the network.

3.3 Model Robustness and Hyperparameter Tuning

In this final section, we demonstrate how VERIFAI can be used to tune test parameters and hyperparameters of AI systems. For the following case studies, we use OpenAI Gym [4], a framework for experimenting with reinforcement learning algorithms.

First, we consider the problem of testing the robustness of a learned controller for a cart-pole, i.e., a cart that balances an inverted pendulum. We trained a neural network to control the cart-pole using Proximal Policy Optimization algorithms [21] with 100k training episodes. We then used VERIFAI to test the robustness of the learned controller, varying the initial lateral position and rotation of the cart as well as the mass and length of the pole. Even for apparently robust controllers, VERIFAI was able to discover configurations for which the cart-pole failed to self-balance. Figure 5 shows 1000 iterations of the falsifier, where sampling was guided by the reward function used by OpenAI to train the controller. This function provides a negative reward if the cart moves more than 2.4 m or if at any time the angle maintained by the pole is greater than 12°. For testing, we slightly modified these thresholds.

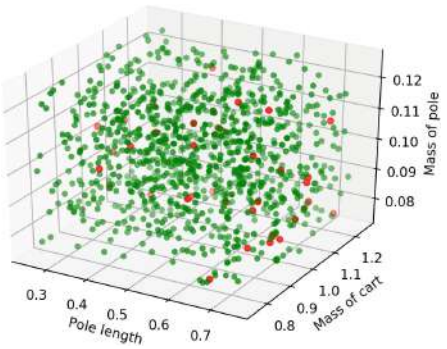


Fig. 5. The green dots represent model parameters for which the cart-pole controller behaved correctly, while the red dots indicate specification violations. Out of 1000 randomly-sampled model parameters, the controller failed to satisfy the specification 38 times. (Color figure online)

Finally, we used VERIFAI to study the effects of hyperparameters when training a neural network controller for a mountain car. In this case, the controller must learn to

exploit momentum in order to climb a steep hill. Here, rather than searching for counterexamples, we look for a set of hyperparameters under which the network *correctly* learns to control the car. Specifically, we explored the effects of using different training algorithms (from a discrete set of choices) and the size of the training set. We used the VERIFAI falsifier to search the hyperparameter space, guided again by the reward function provided by OpenAI Gym (here the distance from the goal position), but negated so that falsification implied finding a controller which successfully climbs the hill. In this way VERIFAI built a table of safe hyperparameters. PCA analysis then revealed which hyperparameters the training process is most sensitive or robust to.

4 Conclusion

We presented VERIFAI, a toolkit for the formal design and analysis of AI/ML-based systems. Our implementation, plus the examples described in Sect. 3, are available in the tool distribution [1], including detailed instructions and expected output.

In future work, we plan to explore additional applications of VERIFAI, and to expand its functionality with new algorithms. Towards the former, we have already interfaced VERIFAI to the CARLA driving simulator [6], for more sophisticated experiments with autonomous cars, as well as to the X-Plane flight simulator [19], for testing an ML-based aircraft navigation system. More broadly, although our focus has been on CPS, we note that VERIFAI's architecture is applicable to other types of systems. Finally, for extending VERIFAI itself, we plan to move beyond directed simulation by incorporating symbolic methods, such as those used in finding adversarial examples.

References

1. VerifAI: a toolkit for the design and analysis of artificial intelligence-based systems. https://github.com/BerkeleyLearnVerify/VerifAI
2. Alur, R., Henzinger, T.A.: Logics and models of real time: a survey. In: de Bakker, J.W., Huizing, C., de Roever, W.P., Rozenberg, G. (eds.) REX 1991. LNCS, vol. 600, pp. 74–106. Springer, Heidelberg (1992). https://doi.org/10.1007/BFb0031988
3. Annpureddy, Y., Liu, C., Fainekos, G.E., Sankaranarayanan, S.: S-taliro: a tool for temporal logic falsification for hybrid systems. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS (2011)
4. Brockman, G., et al.: OpenAI Gym. arXiv:1606.01540 (2016)
5. Butler, M.: Uber's tempe accident raises questions of self-driving safety. East Valley Tribune (2017). http://www.eastvalleytribune.com/local/tempe/uber-s-tempe-accident-raises-questions-of-self-driving-safety/article_30b99e74-189d-11e7-bc1d-07f943301a72.html
6. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: an open urban driving simulator. In: Conference on Robot Learning, CoRL, pp. 1–16 (2017)
7. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. In: Barrett, C., Davies, M., Kahsai, T. (eds.) NFM 2017. LNCS, vol. 10227, pp. 357–372. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57288-8_26
8. Dreossi, T., Donze, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. J. Autom. Reasoning (JAR) (2019)

9. Dreossi, T., Ghosh, S., Yue, X., Keutzer, K., Sangiovanni-Vincentelli, A., Seshia, S.A.: Counterexample-guided data augmentation. In: 27th International Joint Conference on Artificial Intelligence (IJCAI) (2018)
10. Dreossi, T., Jha, S., Seshia, S.A.: Semantic adversarial deep learning. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 3–26. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_1
11. Duggirala, P.S., Mitra, S., Viswanathan, M., Potok, M.: C2E2: a verification tool for stateflow models. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 68–82. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_5
12. Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: a language for scenario specification and scene generation. In: 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI) (2019, to appear)
13. Fremont, D.J., Yue, X., Dreossi, T., Ghosh, S., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: language-based scene generation. CoRR (2018). arXiv:1809.09310
14. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP) (2018)
15. Ghosh, S., Berkenkamp, F., Ranade, G., Qadeer, S., Kapoor, A.: Verifying controllers against adversarial examples with Bayesian optimization. In: 2018 IEEE International Conference on Robotics and Automation (ICRA) (2018)
16. Godefroid, P., Kiezun, A., Levin, M.Y.: Grammar-based whitebox fuzzing. In: ACM SIGPLAN Notices. ACM (2008)
17. Grembek, O., Kurzhanskiy, A.A., Medury, A., Varaiya, P., Yu, M.: Making intersections safer with I2V communication (2019). arXiv:1803.00471, to appear in Transportation Research, Part C
18. Halton, J.H.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. Numer. Math. **2**, 84–90 (1960)
19. Laminar Research: X-Plane 11 (2019). https://www.x-plane.com/
20. Palazzi, A.: Finding lane lines on the road (2018). https://github.com/ndrplz/self-driving-car/tree/master/project_1_lane_finding_basic
21. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR (2017). arXiv:1707.06347
22. Seshia, S.A., et al.: Formal specification for deep neural networks. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 20–34. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_2
23. Seshia, S.A., Sadigh, D., Sastry, S.S.: Towards Verified Artificial Intelligence. CoRR (2016). arXiv:1606.08514
24. Vazquez-Chanlatte, M.: mvcisback/py-metric-temporal-logic: v0.1.1 (2019). https://doi.org/10.5281/zenodo.2548862
25. Webots: Commercial mobile robot simulation software. http://www.cyberbotics.com
26. Wicker, M., Huang, X., Kwiatkowska, M.: Feature-guided black-box safety testing of deep neural networks. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10805, pp. 408–426. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89960-2_22
27. Wu, B., Iandola, F., Jin, P.H., Keutzer, K.: SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In: CVPR 2017 (2016). https://doi.org/10.1109/CVPRW.2017.60

The Marabou Framework for Verification and Analysis of Deep Neural Networks

Guy Katz[1(✉)], Derek A. Huang[2], Duligur Ibeling[2], Kyle Julian[2], Christopher Lazarus[2], Rachel Lim[2], Parth Shah[2], Shantanu Thakoor[2], Haoze Wu[2], Aleksandar Zeljić[2], David L. Dill[2], Mykel J. Kochenderfer[2], and Clark Barrett[2]

[1] The Hebrew University of Jerusalem, Jerusalem, Israel
guykatz@cs.huji.ac.il
[2] Stanford University, Stanford, USA
{huangda,duligur,kjulian3,clazarus,parth95,thakoor, haozewu,zeljic,dill,mykel,clarkbarrett}@stanford.edu, rachelim@cs.stanford.edu

Abstract. Deep neural networks are revolutionizing the way complex systems are designed. Consequently, there is a pressing need for tools and techniques for network analysis and certification. To help in addressing that need, we present *Marabou*, a framework for verifying deep neural networks. Marabou is an SMT-based tool that can answer queries about a network's properties by transforming these queries into constraint satisfaction problems. It can accommodate networks with different activation functions and topologies, and it performs high-level reasoning on the network that can curtail the search space and improve performance. It also supports parallel execution to further enhance scalability. Marabou accepts multiple input formats, including protocol buffer files generated by the popular TensorFlow framework for neural networks. We describe the system architecture and main components, evaluate the technique and discuss ongoing work.

1 Introduction

Recent years have brought about a major change in the way complex systems are being developed. Instead of spending long hours hand-crafting complex software, many engineers now opt to use *deep neural networks* (*DNNs*) [6,19]. DNNs are machine learning models, created by training algorithms that generalize from a finite set of examples to previously unseen inputs. Their performance can often surpass that of manually created software as demonstrated in fields such as image classification [16], speech recognition [8], and game playing [21].

Despite their overall success, the opacity of DNNs is a cause for concern, and there is an urgent need for certification procedures that can provide rigorous guarantees about network behavior. The formal methods community has

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 443–452, 2019.
https://doi.org/10.1007/978-3-030-25540-4_26

taken initial steps in this direction, by developing algorithms and tools for neural network verification [5,9,10,12,18,20,23,24]. A DNN verification query consists of two parts: (i) a neural network, and (ii) a property to be checked; and its result is either a formal guarantee that the network satisfies the property, or a concrete input for which the property is violated (a counter-example). A verification query can encode the fact, e.g., that a network is robust to small adversarial perturbations in its input [22].

A neural network is comprised of *neurons*, organized in layers. The network is evaluated by assigning values to the neurons in the input layer, and then using these values to iteratively compute the assignments of neurons in each succeeding layer. Finally, the values of neurons in the last layer are computed, and this is the network's output. A neuron's assignment is determined by computing a weighted sum of the assignments of neurons from the preceding layer, and then applying to the result a non-linear activation function, such as the Rectified Linear Unit (ReLU) function, $\mathrm{ReLU}(x) = \max(0, x)$. Thus, a network can be regarded as a set of *linear constraints* (the weighted sums), and a set of *non-linear constraints* (the activation functions). In addition to a neural network, a verification query includes a property to be checked, which is given in the form of linear or non-linear constraints on the network's inputs and outputs. The verification problem thus reduces to finding an assignment of neuron values that satisfies all the constraints simultaneously, or determining that no such assignment exists.

This paper presents a new tool for DNN verification and analysis, called *Marabou*. The Marabou project builds upon our previous work on the Reluplex project [2,7,12,13,15,17], which focused on applying SMT-based techniques to the verification of DNNs. Marabou follows the Reluplex spirit in that it applies an SMT-based, *lazy search* technique: it iteratively searches for an assignment that satisfies all given constraints, but treats the non-linear constraints lazily in the hope that many of them will prove irrelevant to the property under consideration, and will not need to be addressed at all. In addition to search, Marabou performs deduction aimed at learning new facts about the non-linear constraints in order to simplify them.

The Marabou framework is a significant improvement over its predecessor, Reluplex. Specifically, it includes the following enhancements and modifications:

- Native support for fully connected and convolutional DNNs with arbitrary piecewise-linear activation functions. This extends the Reluplex algorithm, which was originally designed to support only ReLU activation functions.
- Built-in support for a *divide-and-conquer* solving mode, in which the solver is run with an initial (small) timeout. If the timeout is reached, the solver partitions its input query into simpler sub-queries, increases the timeout value, and repeats the process on each sub-query. This mode naturally lends itself to parallel execution by running sub-queries on separate nodes; however, it can yield significant speed-ups even when used with a single node.
- A complete simplex-based linear programming core that replaces the external solver (GLPK) that was previously used in Reluplex. The new simplex

core was tailored for a smooth integration with the Marabou framework and eliminates much of the overhead in Reluplex due to the use of GLPK.
– Multiple interfaces for feeding queries into the solver. A query's neural network can be provided in a textual format or as a protocol buffer (*protobuf*) file containing a TensorFlow model; and the property can be either compiled into the solver, provided in Python, or stored in a textual format. We expect these interfaces will simplify usage of the tool for many users.
– Support for network-level reasoning and deduction. The earlier Reluplex tool performed deductions at the level of single constraints, ignoring the input network's topology. In Marabou, we retain this functionality but also include support for reasoning based on the network topology, such as symbolic bound tightening [23]. This allows for efficient curtailment of the search space.

Marabou is available online [14] under the permissive modified BSD license.

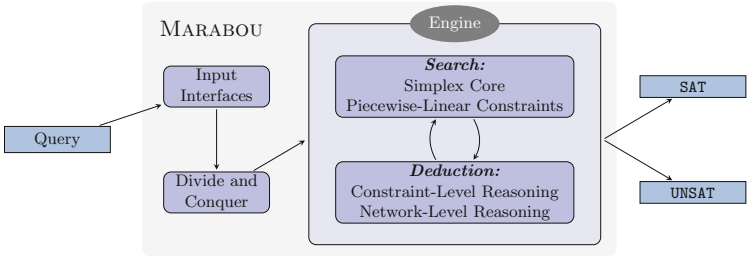


Fig. 1. The main components of Marabou.

2 Design of Marabou

Marabou regards each neuron in the network as a variable and searches for a variable assignment that simultaneously satisfies the query's linear constraints and non-linear constraints. At any given point, Marabou maintains the current variable assignment, lower and upper bounds for every variable, and the set of current constraints. In each iteration, it then changes the variable assignment in order to (1) correct a violated linear constraint, or (2) correct a violated non-linear constraint.

The Marabou verification procedure is sound and complete, i.e. the aforementioned loop eventually terminates. This can be shown via a straightforward extension of the soundness and completeness proof for Reluplex [12]. However, in order to guarantee termination, Marabou only supports activation functions that are piecewise-linear. The tool already has built-in support for the ReLU function and the Max function $\max(x_1, \ldots, x_n)$, and it is modular in the sense that additional piecewise-linear functions can be added easily.

Another important aspect of Marabou's verification strategy is deduction—specifically, the derivation of tighter lower and upper variable bounds. The motivation is that such bounds may transform piecewise-linear constraints into linear constraints, by restricting them to one of their linear segments. To achieve this, Marabou repeatedly examines linear and non-linear constraints, and also performs network-level reasoning, with the goal of discovering tighter variable bounds.

Next, we describe Marabou's main components (see also Fig. 1).

2.1 Simplex Core (*Tableau* and *BasisFactorization* Classes)

The simplex core is the part of the system responsible for making the variable assignment satisfy the linear constraints. It does so by implementing a variant of the *simplex algorithm* [3]. In each iteration, it changes the assignment of some variable x, and consequently the assignment of any variable y that is connected to x by a linear equation. Selecting x and determining its new assignment is performed using standard algorithms—specifically, the *revised simplex method* in which the various linear constraints are kept in implicit matrix form, and the steepest-edge and Harris' ratio test strategies for variable selection.

Creating an efficient simplex solver is complicated. In Reluplex, we delegated the linear constraints to an external solver, GLPK. Our motivation for implementing a new custom solver in Marabou was twofold: first, we observed in Reluplex that the repeated translation of queries into GLPK and extraction of results from GLPK was a limiting factor on performance; and second, a black box simplex solver did not afford the flexibility we needed in the context of DNN verification. For example, in a standard simplex solver, variable assignments are typically pressed against their upper or lower bounds, whereas in the context of a DNN, other assignments might be needed to satisfy the non-linear constraints. Another example is the deduction capability, which is crucial for efficiently verifying a DNN and whose effectiveness might depend on the internal state of the simplex solver.

2.2 Piecewise-Linear Constraints (*PiecewiseLinearConstraint* Class)

Throughout its execution, Marabou maintains a set of piecewise-linear constraints that represent the DNN's non-linear functions. In iterations devoted to satisfying these constraints, Marabou looks for any constraints that are not satisfied by the current assignment. If such a constraint is found, Marabou changes the assignment in a way that makes that constraint satisfied. Alternatively, in order to guarantee eventual termination, if Marabou detects that a certain constraint is repeatedly not satisfied, it may perform a *case-split* on that constraint: a process in which the piecewise-linear constraint φ is replaced by an equivalent disjunction of linear constraints $c_1 \vee \ldots \vee c_n$. Marabou considers these disjuncts one at a time and checks for satisfiability. If the problem is satisfiable when φ is

replaced by some c_i, then the original problem is also satisfiable; otherwise, the original problem is unsatisfiable.

In our implementation, piecewise-linear constraints are represented by objects of classes that inherit from the *PiecewiseLinearConstraint* abstract class. Currently the two supported instances are ReLU and Max, but the design is modular in the sense that new constraint types can easily be added. *PiecewiseLinearConstraint* defines the interface methods that each supported piecewise-linear constraint needs to implement. Some of the key interface methods are:

- *satisfied()*: the constraint object needs to answer whether or not it is satisfied given the current assignment. For example, for a constraint $y = \text{ReLU}(x)$ and assignment $x = y = 3$, *satisfied()* would return *true*; whereas for assignment $x = -5, y = 3$, it would return *false*.
- *getPossibleFixes()*: if the constraint is not satisfied by the current assignment, this method returns possible changes to the assignment that would correct the violation. For example, for $x = -5, y = 3$, the ReLU constraint from before might propose two possible changes to the assignment, $x \leftarrow 3$ or $y \leftarrow 0$, as either would satisfy $y = \text{ReLU}(x)$.
- *getCaseSplits()*: this method asks the piecewise-linear constraint φ to return a list of linear constraints $c_1, \ldots, c_n$, such that φ is equivalent to $c_1 \vee \ldots \vee c_n$. For example, when invoked for a constraint $y = \max(x_1, x_2)$, *getCaseSplits()* would return the linear constraints $c_1 : (y = x_1 \wedge x_1 \geq x_2)$ and $c_2 : (y = x_2 \wedge x_2 \geq x_1)$. These constraints satisfy the requirement that the original constraint is equivalent to $c_1 \vee c_2$.
- *getEntailedTightenings()*: as part of Marabou's deduction of tighter variable bounds, piecewise-linear constraints are repeatedly informed of changes to the lower and upper bounds of variables they affect. Invoking *getEntailedTightenings()* queries the constraint for tighter variable bounds, based on current information. For example, suppose a constraint $y = \text{ReLU}(x)$ is informed of the upper bounds $x \leq 5$ and $y \leq 7$; in this case, *getEntailedTightenings()* would return the tighter bound $y \leq 5$.

2.3 Constraint- and Network-Level Reasoning (*RowBoundTightener*, *ConstraintBoundTightener* and *SymbolicBoundTightener* Classes)

Effective deduction of tighter variable bounds is crucial for Marabou's performance. Deduction is performed at the constraint level, by repeatedly examining linear and piecewise-linear constraints to see if they imply tighter variable bounds; and also at the DNN-level, by leveraging the network's topology.

Constraint-level bound tightening is performed by querying the piecewise-linear constraints for tighter bounds using the *getEntailedTightenings()* method. Similarly, linear equations can also be used to deduce tighter bounds. For example, the equation $x = y + z$ and lower bounds $x \geq 0$, $y \geq 1$ and $z \geq 1$ together imply the tighter bound $x \geq 2$. As part of the simplex-based search, Marabou repeatedly encounters many linear equations and uses them for bound tightening.

Several recent papers have proposed verification schemes that rely on DNN-level reasoning [5,23]. Marabou supports this kind of reasoning as well, by storing the initial network topology and performing deduction steps that use this information as part of its iterative search. DNN-level reasoning is seamlessly integrated into the search procedure by (1) initializing the DNN-level reasoners with the most up-to-date information discovered during the search, such as variable bounds and the state of piecewise-linear constraints; and (2) feeding any new information that is discovered back into the search procedure. Presently Marabou implements a symbolic bound tightening procedure [23]: based on network topology, upper and lower bounds for each hidden neuron are expressed as a linear combination of the input neurons. Then, if the bounds on the input neurons are sufficiently tight (e.g., as a result of past deductions), these expressions for upper and lower bounds may imply that some of the hidden neurons' piecewise-linear activation functions are now restricted to one of their linear segments. Implementing additional DNN-level reasoning operations is work in progress.

2.4 The Engine (*Engine* and *SmtCore* Classes)

The main class of Marabou, in which the main loop resides, is called the *Engine*. The engine stores and coordinates the various solution components, including the simplex core and the piecewise-linear constraints. The main loop consists, roughly, of the following steps (the first rule that applies is used):

1. If a piecewise-linear constraint had to be fixed more than a certain number of times, perform a case split on that constraint.
2. If the problem has become unsatisfiable, e.g. because for some variable a lower bound has been deduced that is greater than its upper bound, undo a previous case split (or return `UNSAT` if no such case split exists).
3. If there is a violated linear constraint, perform a simplex step.
4. If there is a violated piecewise-linear constraint, attempt to fix it.
5. Return `SAT` (all constraints are satisfied).

The engine also triggers deduction steps, both at the neuron level and at the network level, according to various heuristics.

2.5 The Divide-and-Conquer Mode and Concurrency (*DnC.py*)

Marabou supports a *divide-and-conquer* ($D\&C$) solving mode, in which the input region specified in the original query is partitioned into sub-regions. The desired property is checked on these sub-regions independently. The D&C mode naturally lends itself to parallel execution, by having each sub-query checked on a separate node. Moreover, the D&C mode can improve Marabou's overall performance even when running sequentially: the total time of solving the sub-queries is often less than the time of solving the original query, as the smaller input regions allow for more effective deduction steps.

Given a query ϕ, the solver maintains a queue Q of $\langle$query, timeout$\rangle$ pairs. Q is initialized with one element $\langle \phi, T \rangle$, where T, the initial timeout, is a configurable parameter. To solve ϕ, the solver loops through the following steps:

1. Pop a pair $\langle \phi', t' \rangle$ from Q and attempt to solve ϕ' with a timeout of t'.
2. If the problem is `UNSAT` and Q is empty, return `UNSAT`.
3. If the problem is `UNSAT` and Q is not empty, return to step 1.
4. If the problem is `SAT`, return `SAT`.
5. If a timeout occurred, split ϕ' into k sub-queries $\phi'_1, \ldots, \phi'_k$ by partitioning its input region. For each sub-query ϕ'_i, push $\langle \phi'_i, m \cdot t' \rangle$ into Q.

The timeout factor m and the splitting factor k are configurable parameters. Splitting the query's input region is performed heuristically.

2.6 Input Interfaces (*AcasParser* class, *maraboupy* Folder)

Marabou supports verification queries provided through the following interfaces:

- Native Marabou format: a user prepares a query using the Marabou C++ interface, compiles the query into the tool, and runs it. This format is useful for integrating Marabou into a larger framework.
- Marabou executable: a user runs a Marabou executable, and passes to it command-line parameters indicating the network and property files to be checked. Currently, network files are encoded using the *NNet* format [11], and the properties are given in a simple textual format.
- Python/TensorFlow interface: the query is passed to Marabou through Python constructs. The python interface can also handle DNNs stored as TensorFlow protobuf files.

3 Evaluation

For our evaluation we used the ACAS Xu [12], CollisionDetection [4] and TwinStream [1] families of benchmarks. Tool-wise, we considered the Reluplex tool which is the most closely related to Marabou, and also ReluVal [23] and Planet [4]. The version of Marabou used for the evaluation is available online [14].

The top left plot in Fig. 3 compares the execution times of Marabou and Reluplex on 180 ACAS Xu benchmarks with a 1 hour timeout. We used Marabou in D&C mode with 4 cores and with $T = 5$, $k = 4$, and $m = 1.5$. The remaining three plots depict an execution time comparison between Marabou D&C (configuration as above), ReluVal and Planet, using 4 cores and a 1 hour timeout. Marabou and Reluval are evaluated over 180 ACAS Xu benchmarks (top right plot), and Marabou and Planet are evaluated on those 180 benchmarks (bottom left plot) and also on 500 CollisionDetection and 81 TwinStream benchmarks (bottom right plot). Due to technical difficulties, ReluVal was not run on the CollisionDetection and TwinStream benchmarks. The results show that in a 4 cores setting Marabou generally outperforms Planet, but generally does not outperform ReluVal (though it does better on some benchmarks). These results highlight the need for additional DNN-level reasoning in Marabou, which is a key ingredient in ReluVal's verification procedure.

Figure 2 shows the average runtime of Marabou and ReluVal on the ACAS Xu properties, as a function of the number of available cores. We see that as the number of cores increases, Marabou (solid) is able to close the gap, and sometimes outperform, ReluVal (dotted). With 64 cores, Marabou outperforms ReluVal on average, and both solvers were able to solve all ACAS Xu benchmarks within 2 hours (except for a few segfaults by ReluVal).

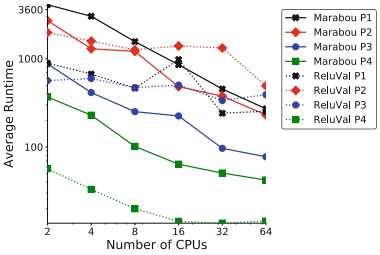


Fig. 2. A scalability comparison of Marabou and ReluVal on ACAS Xu.

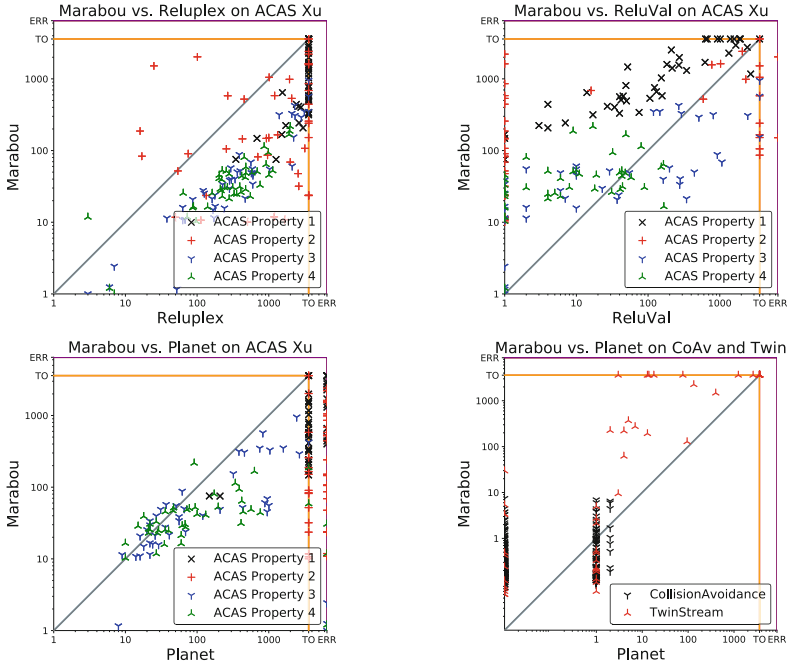


Fig. 3. A comparison of Marabou with Reluplex, ReluVal and Planet.

4 Conclusion

DNN analysis is an emerging field, and Marabou is a step towards a more mature, stable verification platform. Moving forward, we plan to improve Marabou in several dimensions. Part of our motivation in implementing a custom simplex solver was to obtain the needed flexibility for fusing together the solving process for linear and non-linear constraints. Currently, this flexibility has not been leveraged much, as these pieces are solved relatively separately. We expect that by tackling

both kinds of constraints simultaneously, we will be able to improve performance significantly. Other enhancements we wish to add include: additional network-level reasoning techniques based on abstract interpretation; better heuristics for both the linear and non-linear constraint solving engines; and additional engineering improvements, specifically within the simplex engine.

Acknowledgements. We thank Elazar Cohen, Justin Gottschlich, and Lindsey Kuper for their contributions to this project. The project was partially supported by grants from the Binational Science Foundation (2017662), the Defense Advanced Research Projects Agency (FA8750-18-C-0099), the Federal Aviation Administration, Ford Motor Company, Intel Corporation, the Israel Science Foundation (683/18), the National Science Foundation (1814369, DGE-1656518), Siemens Corporation, and the Stanford CURIS program.

References

1. Bunel, R., Turkaslan, I., Torr, P., Kohli, P., Kumar, M.: Piecewise linear neural network verification: a comparative study. Technical report (2017). arXiv:1711.00455v1
2. Carlini, N., Katz, G., Barrett, C., Dill, D.: Provably minimally-distorted adversarial examples. Technical report (2017). arXiv:1709.10207
3. Chvátal, V.: Linear Programming. W. H. Freeman and Company, New York (1983)
4. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
5. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, E., Chaudhuri, S., Vechev, M.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: Proceedings of 39th IEEE Symposium on Security and Privacy (S&P) (2018)
6. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
7. Gopinath, D., Katz, G., Păsăreanu, C., Barrett, C.: DeepSafe: a data-driven approach for checking adversarial robustness in neural networks. In: Proceedings of 16th International Symposium on on Automated Technology for Verification and Analysis (ATVA), pp. 3–19 (2018)
8. Hinton, G., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
9. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Proceedings of 29th International Conference on Computer Aided Verification (CAV), pp. 3–29 (2017)
10. Hull, J., Ward, D., Zakrzewski, R.: Verification and validation of neural networks for safety-critical applications. In: Proceedings of 21st American Control Conference (ACC) (2002)
11. Julian, K.: NNet Format (2018). https://github.com/sisl/NNet
12. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5

13. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Towards proving the adversarial robustness of deep neural networks. In: Proceedings of 1st Workshop on Formal Verification of Autonomous Vehicles (FVAV), pp. 19–26 (2017)
14. Katz, G., et al.: Marabou (2019). https://github.com/guykatzz/Marabou/tree/cav_artifact
15. Kazak, Y., Barrett, C., Katz, G., Schapira, M.: Verifying deep-RL-driven systems. In: Proceedings of 1st ACM SIGCOMM Workshop on Network Meets AI & ML (NetAI) (2019)
16. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
17. Kuper, L., Katz, G., Gottschlich, J., Julian, K., Barrett, C., Kochenderfer, M.: Toward scalable verification for safety-critical deep networks. Technical report (2018). arXiv:1801.05950
18. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 243–257. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_24
19. Riesenhuber, M., Tomaso, P.: Hierarchical models of object recognition in cortex. Nat. Neurosci. **2**(11), 1019–1025 (1999)
20. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: Proceedings of 27th International Joint Conference on Artificial Intelligence (IJCAI) (2018)
21. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
22. Szegedy, C., et al.: Intriguing properties of neural networks. Technical report (2013). arXiv:1312.6199
23. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. Technical report (2018). arXiv:1804.10829
24. Xiang, W., Tran, H., Johnson, T.: Output reachable set estimation and verification for multi-layer neural networks. IEEE Trans. Neural Netw. Learn. Syst. (TNNLS) **99**, 1–7 (2018)

Probabilistic Systems, Runtime Techniques

Probabilistic Bisimulation for Parameterized Systems
(with Applications to Verifying Anonymous Protocols)

Chih-Duo Hong[1(✉)], Anthony W. Lin[2(✉)], Rupak Majumdar[3(✉)], and Philipp Rümmer[4(✉)]

[1] Oxford University, Oxford, UK
chihduo.hong@gmail.com
[2] TU Kaiserslautern, Kaiserslautern, Germany
lin@cs.uni-kl.de
[3] Max Planck Institute for Software Systems, Kaiserslautern, Germany
rupak@mpi-sws.org
[4] Uppsala University, Uppsala, Sweden
philipp.ruemmer@it.uu.se

Abstract. Probabilistic bisimulation is a fundamental notion of process equivalence for probabilistic systems. It has important applications, including the formalisation of the anonymity property of several communication protocols. While there is a large body of work on verifying probabilistic bisimulation for finite systems, the problem is in general undecidable for parameterized systems, i.e., for infinite families of finite systems with an arbitrary number n of processes. In this paper we provide a general framework for reasoning about probabilistic bisimulation for parameterized systems. Our approach is in the spirit of software verification, wherein we encode proof rules for probabilistic bisimulation and use a decidable first-order theory to specify systems and candidate bisimulation relations, which can then be checked automatically against the proof rules.

We work in the framework of regular model checking, and specify an infinite-state system as a regular relation described by a first-order formula over a universal automatic structure, i.e., a logical theory over the string domain. For probabilistic systems, we show how probability values (as well as the required operations) can be encoded naturally in the logic. Our main result is that one can specify the verification condition of whether a given regular binary relation is a probabilistic bisimulation as a regular relation. Since the first-order theory of the universal automatic structure is decidable, we obtain an effective method for verifying probabilistic bisimulation for infinite-state systems, given a regular relation as a candidate proof. As a case study, we show that our framework is sufficiently expressive for proving the anonymity property of the parameterized dining cryptographers protocol and the parameterized grades protocol. Both of these protocols hitherto could not be verified by existing automatic methods.

This research was sponsored in part by the ERC Starting Grant 759969 (AV-SMP), ERC Synergy project 610150 (ImPACT), the DFG project 389792660-TRR 248 (Perspicuous Computing), the Swedish Research Council (VR) under grant 2018-04727, and by the Swedish Foundation for Strategic Research (SSF) under the project WebSec (Ref. RIT17-0011).

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 455–474, 2019.
https://doi.org/10.1007/978-3-030-25540-4_27

Moreover, with the help of standard automata learning algorithms, we show that the candidate relations can be synthesized fully automatically, making the verification fully automated.

1 Introduction

Equivalence checking using bisimulation relations plays a fundamental role in formal verification. Bisimulation is the basis for substitutability of systems: if two systems are bisimilar, their behaviors are the same and they satisfy the same formulas in expressive temporal logics. The notion of bisimulation is defined both for deterministic [39] and for probabilistic transition systems [34]. In both contexts, checking bisimulation has many applications, such as proving correctness of anonymous communication protocols [15], reasoning about knowledge [22], program optimization [32], and optimizations for computational problems (e.g. language equivalence and minimization) of finite automata [12].

The problem of checking bisimilarity of two given systems has been widely studied. It is decidable in polynomial-time for both probabilistic and non-probabilistic *finite-state* systems [6,17,20,52]. These algorithms form the basis of practical tools for checking bisimulation. For infinite-state systems, such as parameterized versions of communication protocols (i.e. infinite families of finite-state systems with an arbitrary number n of processes), the problem is undecidable in general. Most research hitherto has focused on identifying decidable subcases (e.g. strong bisimulations for pushdown systems for probabilistic and non-probabilistic cases [25,47,48]), rather than on providing tool support for practical problems.

In this paper, we propose a first-order verification approach—inspired by software verification techniques—for reasoning about bisimilarity for infinite-state systems. In our approach, we provide first-order logic *proof rules* to determine if a given binary relation is a bisimulation. To this end, we must find an *encoding* of systems and relations and a *decidable first-order theory* that can formalize the system, the property, and the proof rules. We propose to use the decidable first-order theory of the *universal automatic structure* [8,10]. Informally, the domain of the theory is a set of words over a finite alphabet Σ, and it captures the first-order theory of the infinite $|\Sigma|$-ary tree with a relation that relates strings of the same level. The theory can express precisely the class of all *regular relations* [8] (a.k.a. automatic relations [10]), which are relations $\varphi(x_1, \ldots, x_k)$ over strings Σ^* that can be recognized by synchronous multi-tape automata. It is also sufficiently powerful to capture many classes of non-probabilistic infinite-state systems and regular model checking [3,13,49–51].

We demonstrate the effectiveness of the approach by encoding and automatically verifying some challenging examples from the literature of parameterized systems in our logic: the anonymity property of the parameterized dining cryptographers protocol [16] and the grades protocol [29]. These examples were only automatically verified for some fixed parameters using finite-state model checkers or equivalence checkers (e.g. see [28,29]). Just as invariant verification for software separates out the proof rules (verification conditions in a decidable logic) from the synthesis of invariants, we separate out proof rules for bisimulation from the synthesis of bisimulation relations.

We demonstrate how recent developments in generating and refining candidate proofs as automata (e.g. [18,26,27,37,38,40,41,53]) can be used to automate the search of proofs, making our verification fully "push button."

Contributions. Our contributions are as follows. First, we show how probabilistic infinite-state systems can be faithfully encoded in the first-order theory of universal automatic structure. In the past, the theory has been used to reason about qualitative liveness of weakly-finite MDPs (e.g. see [36,37]), which allows the authors to disregard the actual non-zero probability values. To the best of our knowledge, no encoding of probabilistic transition systems in the theory was available. In order to be able to effectively encode probabilistic systems, our theory should typically be two-sorted: one sort for encoding the configurations, and the other for encoding the probability values. We show how both sorts (and the operations required for the sorts) can be encoded in the universal automatic structure, which requires only the domain of strings. In the sequel, such transition systems will be called *regular transition systems*.

Second, using the *minimal probability assumption* on the transition systems [34] (i.e. there exists an $\varepsilon > 0$ such that any non-zero transition probability is at least ε)—which is often satisfied in practice—we show how the verification condition of whether a given regular binary relation is a probabilistic bisimulation can be encoded in the theory. The decidability of the first-order theory over the universal automatic structure gives us an effective means of checking probabilistic bisimulation for regular transition systems. In fact, the theory can be easily reduced to the weak monadic theory WS1S of one successor (therefore, allowing highly optimized tools like Mona [31] and Gaston [23]) by interpreting finite words as finite sets (e.g. see [19,46]).

Our framework requires the encoding of the systems and the proofs in the first-order theory of the universal automatic structure. Which interesting examples can it capture? Our third contribution is to provide two examples from the literature of parameterized verification: the anonymity property of the parameterized dining cryptographers protocol [16] and of the parameterized grades protocol [29]. We study two versions of dining cryptographers protocol in this paper: the classical version where the secrets are single bits, and a generalized version where the secrets are bit-vectors of arbitrary length.

Thus far, our framework requires a candidate proof to be supplied by the user. Our final contribution is to demonstrate how standard techniques from the synthesis literature (e.g. automata learning [18,26,27,37,38,40,41,53]) can be used to fully automate the proof search. Using automata learning, we successfully pinpoint regular proofs for the anonymity property of the three protocols: the two dining cryptographers protocols are verified in 6 and 28 s, respectively, and the grades protocol in 35 s.

Other Related Work. The verification framework we use in this paper can be construed as a regular model checking [3] framework using regular relations. The framework uses first-order logic as the language, which makes it convenient to express many verification conditions (as is well-known from first-order theorem proving [14]). The use of the universal automatic structure allows us to express two different sorts (configurations and probability values) in one sort (i.e. strings). Most work in regular model checking focuses on safety and liveness properties (e.g. [2,3,11,13,27,36,37,40,42,49,51,53]).

Some automated techniques can prove the anonymity property of the dining cryptographers protocol and the grades protocol in the finite case, e.g., the PRISM model

checker [28,45] and language equivalence by the tool APEX [29]. To the best of our knowledge, our method is the first automated technique proving the anonymity property of the protocols in the parameterized case.

Our work is in spirit of deductive software verification (e.g., [4,14,24,35,43,44]), where one provides inductive invariants manually, and a tool automatically checks correctness of the candidate invariants. In theory, our result yields a fully-automatic procedure by enumerating all candidate regular proofs, and at the same time enumerating all candidate counterexamples (note that we avoid undecidability by restricting attention to proofs encodable as regular relations). In our implementation, we use recent advances in automata-learning based synthesis to efficiently encode the search [18,37].

2 Preliminaries

General Notation. We use $\mathbb{N}$ to denote non-negative integers. Given $a, b \in \mathbb{R}$, we use a standard notation $[a, b] := \{c \in \mathbb{R} : a \leq c \leq b\}$ to denote real intervals. Given a set S, we use S^* to denote the set of all finite sequences of elements from S. The set S^* always includes the empty sequence which we denote by ε. We call a function $f : S \to [0, 1]$ a *probability distribution over* S if $\sum_{s \in S} f(s) = 1$. We shall use I_s to denote the probability distribution f with $f(s) = 1$, and $\mathcal{D}_S$ to denote the set of probability distributions over S. Given a function $f : X_1 \times \cdots \times X_n \to Y$, the *graph* of f is the relation $\{(x_1, ..., x_n, f(x_1, ..., x_n)) : \forall i \in \{1, \ldots, n\}.\ x_i \in X_i\}$. Whenever a relation R is an equivalence relation over set S, we use S/R to denote the set of equivalence classes created by R. Depending on the context, we may use $p\, R\, q$ or $R(p, q)$ to denote $(p, q) \in R$.

Words and Automata. We assume basic familiarity with word automata. Fix a finite alphabet Σ. For each finite word $w := w_1 \ldots w_n \in \Sigma^*$, we write $w[i, j]$, where $1 \leq i \leq j \leq n$, to denote the segment $w_i \ldots w_j$. Given an automaton $\mathcal{A} := (\Sigma, Q, \delta, q_0, F)$, a run of $\mathcal{A}$ on w is a function $\rho : \{0, \ldots, n\} \to Q$ with $\rho(0) = q_0$ that obeys the transition relation δ. We may also denote the run ρ by the word $\rho(0) \cdots \rho(n)$ over the alphabet Q. The run ρ is said to be *accepting* if $\rho(n) \in F$, in which case we say that the word w is *accepted* by $\mathcal{A}$. The language $L(\mathcal{A})$ of $\mathcal{A}$ is the set of words in Σ^* accepted by $\mathcal{A}$.

Transition Systems. We fix a set ACT of *action symbols*. A *transition system* over ACT is a tuple $\mathfrak{S} := \langle S; \{\rightarrow_a\}_{a \in \mathsf{ACT}} \rangle$, where S is a set of *configurations* and $\rightarrow_a \subseteq S \times S$ is a binary relation over S. We use $\rightarrow$ to denote the relation $\bigcup_{a \in \mathsf{ACT}} \rightarrow_a$. We say that a sequence $s_1 \rightarrow \cdots \rightarrow s_{n+1}$ is a *path* in $\mathfrak{S}$ if $s_1, ..., s_{n+1} \in S$ and $s_i \rightarrow s_{i+1}$ for $i \in \{1, \ldots, n\}$. A transition system is called *bounded branching* if the number of configurations reachable from a configuration in one step is bounded. Formally, this means that there exists an *a priori* integer N such that for all $s \in S$, $|\{s' \in S : s \rightarrow s'\}| \leq N$.

Probabilistic Transition Systems. A *probabilistic transition system* (*PTS*) [34] is a structure $\mathfrak{S} := \langle S; \{\delta_a\}_{a \in \mathsf{ACT}} \rangle$ where S is a set of configurations and $\delta_a : S \to \mathcal{D}_S \cup \{\overline{0}\}$ maps each configuration to either a probability distribution or a zero function $\overline{0}$. Here $\delta_a(s) = \overline{0}$ simply means that s is a "dead end" for action a. We shall use $\delta_a(s, s')$ to denote $\delta_a(s)(s')$. In this paper, we always assume that $\delta_a(s, s')$ is a rational number

and $|\{s' : \delta_a(s,s') \neq 0\}| < \infty$. The *underlying transition graph* of a PTS is a transition system $\langle S; \{\rightarrow_a\}_{a\in\mathsf{ACT}}\rangle$ such that $s \rightarrow_a s'$ iff $\delta_a(s,s') \neq 0$.

It is standard (e.g. see [34]) to impose the *minimal probability assumption* on the PTS that we shall be dealing with, i.e., there is $\epsilon > 0$ such that any transition with a non-zero probability p satisfies $p > \epsilon$. This assumption is practically sensible since it is satisfied by most PTSs that we deal with in practice (e.g. finite PTS, probabilistic pushdown automata [21], and most examples from probabilistic parameterized systems [36,37] including our examples from Sect. 5). The minimal probability assumption, among others, implies that the PTS is bounded-branching (i.e. that its underlying transition graph is bounded-branching). In the sequel, we shall adopt this assumption.

Probabilistic Bisimulations. Let $\mathfrak{S} := \langle S; \{\delta_a\}_{a\in\mathsf{ACT}}\rangle$ be a PTS. We write $s \xrightarrow{\rho}_a S'$ if $\sum_{s'\in S'} \delta_a(s,s') = \rho$. A *probabilistic bisimulation* for $\mathfrak{S}$ is an equivalence relation R over S, such that $(p,q) \in R$ implies

$$\forall a \in \mathsf{ACT}.\ \forall S' \in S/R.\ (p \xrightarrow{\rho}_a S' \Leftrightarrow q \xrightarrow{\rho}_a S'). \tag{1}$$

We say that p and q are *probabilistic bisimilar* (written as $p \sim q$) if there is a probabilistic bisimulation R such that $(p,q) \in R$. We can compute probabilistic bisimulation between two PTSs $\mathfrak{S} := \langle S; \{\delta_a\}_{a\in\mathsf{ACT}}\rangle$ and $\mathfrak{S}' := \langle S'; \{\delta'_a\}_{a\in\mathsf{ACT}}\rangle$ by computing a probabilistic bisimulation R for the disjoint union of $\mathfrak{S}$ and $\mathfrak{S}'$, which is defined as $\mathfrak{S} \sqcup \mathfrak{S}' := \langle S \sqcup S'; \{\delta''_a\}_{a\in\mathsf{ACT}}\rangle$ where $\delta''_a(s) := \delta_a(s)$ for $s \in S$, and $\delta''_a(s) := \delta'_a(s)$ for $s \in S'$. In such case, we say R is a probabilistic bisimulation between $\mathfrak{S}$ and $\mathfrak{S}'$.

3 Framework of Regular Relations

In this section we describe the framework of regular relations for specifying probabilistic infinite-state systems, properties to verify, and proofs, all in a uniform symbolic way. The framework is amenable to automata-theoretic algorithms in the spirit of *regular model checking* [3,13].

The framework of *regular relations* [8] (a.k.a. *automatic relations* [9]) uses the first-order theory of universal[1] automatic structure

$$\mathfrak{U} := \langle \Sigma^*; \preceq, \mathrm{eqL}, \{l_a\}_{a\in\Sigma}\rangle, \tag{2}$$

where Σ is some finite alphabet, $\preceq$ is the (non-strict) prefix-of relation, eqL is the binary equal length predicate, and l_a is a unary predicate asserting that the last letter of the word is a. The domain of the structure is the set of finite words over Σ, and for words $w, w' \in \Sigma^*$, we have $w \preceq w'$ iff there is some $w'' \in \Sigma^*$ such that $w \cdot w'' = w'$, $\mathrm{eqL}(w,w')$ iff $|w| = |w'|$, and $l_a(w)$ iff there is some $w'' \in \Sigma^*$ such that $w = w'' \cdot a$.

Next, we discuss the expressive power of first-order formulas over the universal automatic structures, and decision procedures for satisfiability of such formulas. In Sect. 4, we shall describe: (1) how to specify a PTS as a first-order formula in $\mathfrak{U}$, and (2) how to specify the verification condition for probabilistic bisimulation property in this theory. In Sect. 5, we shall show that the theory is sufficiently powerful for capturing probabilistic bisimulations for interesting examples.

[1] Here, "universal" simply means that all automatic structures are first-order interpretable in this structure.

Expressiveness and Decidability. The name "regular" associated with this framework is because the set of formulas $\varphi(x_1, \ldots, x_k)$ first-order definable in $\mathfrak{U}$ coincides with *regular relations*, i.e., relations definable by synchronous automata. More precisely, we define $[\![\varphi]\!]$ as the relation which contains all tuples $(w_1, \ldots, w_k) \in (\Sigma_\perp^*)^k$ such that $\mathfrak{U} \models \varphi(w_1, \ldots, w_k)$. In addition, we define the *convolution* $w_1 \otimes \cdots \otimes w_k$ of words $w_1, \ldots, w_k \in \Sigma^*$ as a word w over $\Sigma_\perp^k$ (where $\perp \notin \Sigma$) such that $w[i] = (a_1, \ldots, a_k)$ with

$$a_j = \begin{cases} w_j[i] & \text{if } |w_j| \geq i, \text{ or} \\ \perp & \text{otherwise.} \end{cases}$$

In other words, w is obtained by juxtaposing $w_1, \ldots, w_k$ and padding the shorter words with $\perp$. For example, $010 \otimes 00 = (0,0)(1,0)(0,\perp)$. A k-ary relation R over Σ^* is *regular* if the set $\{w_1 \otimes \cdots \otimes w_k : (w_1, \ldots, w_k) \in R\}$ is a regular language over the alphabet $\Sigma_\perp^k$. The relationship between $\mathfrak{U}$ and regular relations can be formally stated as follows.

Proposition 1 ([8–10]).

1. *Given a formula $\varphi(\bar{x})$ over $\mathfrak{U}$, the relation $[\![\varphi]\!]$ is effectively regular. Conversely, given a regular relation R, we can compute a formula $\varphi(\bar{x})$ over $\mathfrak{U}$ such that $[\![\varphi]\!] = R$.*
2. *The first-order theory of $\mathfrak{U}$ is decidable.*

The decidability of the first-order theory of $\mathfrak{U}$ follows using a standard automata-theoretic algorithm (e.g. see [9,49]).

In the sequel, we shall also use the term regular relations to denote relations definable in $\mathfrak{U}$. In addition, to avoid notational clutter, we shall freely use other regular relations (e.g. successor relation $\prec_{succ}$ of the prefix $\preceq$, and membership in a regular language) as syntactic sugar.

We note that the first-order theory of $\mathfrak{U}$ can also be reduced to weak monadic theory WS1S of one successor (therefore, allowing highly optimized tools like MONA [31] and Gaston [23]) by translating finite words to finite sets. The relationship between the universal automatic structure and WS1S can be made precise using the notion of *finite-set interpretations* [19,46].

4 Probabilistic Bisimilarity Within Regular Relations

In this section, we show how the framework of regular relations can be used to encode a PTS, and the corresponding proof rules for probabilistic bisimulation.

4.1 Specifying a Probabilistic Transition System

Since we assume that all probability values specified in our systems are rational numbers, the fact that our PTS is bounded-branching implies that we can specify the probability values by natural *weights* (by multiplying the probability values by the least common multiple of the denominators). For example, if a configuration c has an action

$toss$ that takes it to c_1 and c_2, each with probability 1/2, then the new system simply changes both values of 1/2 to 1. This is a known trick in the literature of probabilistic verification (e.g. see [1]). Therefore, we can now assume that the transition probability functions have range $\mathbb{N}$. *The challenge now is that our encoding of a PTS in the universal automatic structure must encode two different sorts* as words over a finite alphabet Σ: configurations and natural weights.

Now we are ready to show how to specify a PTS $\mathfrak{S}$ in our framework. Fix a finite alphabet Σ containing at least two letters 0 and 1. We encode the domain of $\mathfrak{S}$ as words over Σ. In addition, a natural weight $n \in \mathbb{N}$ can be encoded in the usual way as a binary string. This motivates the following definition.

Definition 1. *Let $\mathfrak{S}$ be a PTS $\langle S; \{\delta_a\}_{a\in\mathsf{ACT}}\rangle$. We say that $\mathfrak{S}$ is regular if the domain S is a regular subset of Σ^* (i.e. definable by a first-order formula $\varphi(x)$ with one free variable over $\mathfrak{U}$), and if the graph of each function δ_a is a ternary regular relation (i.e. definable by a first-order formula $\varphi(x, y, z)$ over $\mathfrak{U}$, where x and y encode configurations, and z encodes a natural weight).*

Definition 1 is quite general since it allows for an infinite number of different natural weights in the PTS. Note that we can make do without the second sort (of numeric weights) if we have only finitely many numeric weights $n_1, \ldots, n_m$. This can be achieved by specifying a regular relation $R_{a,i}$ for each action label $a \in \mathsf{ACT}$ and numeric weight n_i with $i \in \{1, \ldots, m\}$.

Example 1. We show a regular encoding of a very simple PTS: a random walk on the set of natural numbers. At each position x, the system can non-deterministically choose to loop or to move. If the system chooses to loop, it will stay at the same position with probability 1. If the system chooses to move, it will move to $x + 1$ with probability 1/4, or move to $\max(0, x - 1)$ with probability 3/4. Normalising the probability values by multiplying by 4, we obtain the numeric weights of 4, 1, and 3 for the aforementioned transitions, respectively.

To represent the system by regular relations, we encode the positions in unary and the numeric weights in binary. The set of configurations is the regular language 1^*. The graph of the transition probability function can be described by a first-order formula $\varphi(x, y, z) := \varphi_{\text{loop}}(x, y, z) \vee \varphi_{\text{move}}(x, y, z)$ over $\mathfrak{U}$, where

$$\varphi_{\text{loop}}(x, y, z) := x \in 1^* \wedge y \in 1^* \wedge ((x = y \wedge z = 100) \vee (x \neq y \wedge z = 0));$$
$$\begin{aligned}\varphi_{\text{move}}(x, y, z) := x \in 1^* \wedge y \in 1^* \wedge ((x \prec_{succ} y \wedge z = 1) \vee \\ (y \prec_{succ} x \wedge z = 11) \vee (x = \varepsilon \wedge y = \varepsilon \wedge z = 11) \vee \\ (\neg(x \prec_{succ} y) \wedge \neg(y \prec_{succ} x) \wedge \neg(x = \varepsilon \wedge y = \varepsilon) \wedge z = 0)).\end{aligned}$$

□

Example 2. As a second example, consider a PTS (from [25], Example 1) described by a probabilistic pushdown automaton with states $Q = \{p, q, r\}$ and stack symbols $\Gamma = \{X, X', Y, Z\}$. There is a unique action a, and the transition rules δ_a are as follows:

$$\begin{array}{llll} pX \xrightarrow{0.5} qXX & pX \xrightarrow{0.5} p & qX \xrightarrow{1} pXX & rY \xrightarrow{1} rXX \\ rX \xrightarrow{0.3} rYX & rX \xrightarrow{0.2} rYX' & rX \xrightarrow{0.5} r & \\ rX' \xrightarrow{0.4} rYX & rX' \xrightarrow{0.1} rYX' & rX' \xrightarrow{0.5} r & \end{array}$$

A configuration of the PTS is a word in $Q\Gamma^*$, consisting of a state in Q and a word over the stack symbols. A transition can be applied if the prefix of the configuration matches the left hand side of the transition rules above. We encode the PTS as follows: the set of configurations is $Q\Gamma^*$, the weights are represented in binary after normalization, and the transition relation $\varphi(x, y, z)$ encodes the transition rules in disjunction. For example, the disjunct corresponding to the rule $pX \xrightarrow{0.5} qXX$ is

$$x \in Q\Gamma^* \wedge y \in Q\Gamma^* \wedge (\exists u.\, x = pXu \wedge y = qXXu) \wedge z = 101.$$

Note that the PTS is bounded branching with a bound 3. □

4.2 Proof Rules for Probabilistic Bisimulation

Fix the set ACT of action symbols and the branching bound $N \geq 1$, owing to the minimal probability assumption. Consider a two-sorted vocabulary $\sigma = \langle \{P_a\}_{a \in \mathsf{ACT}}, R, + \rangle$, where P_a is a ternary relation (with the first two arguments over the first sort, and the third argument over the second sort of natural numbers), R is a binary relation over the first sort, and $+$ is the addition function over the second sort of natural numbers. The main result we shall show next is summarized in the following theorem:

Theorem 1. *There is a fixed first-order formula Φ over σ such that a binary relation R is a probabilistic bisimulation over a bounded-branching PTS $\mathfrak{S} = \langle S; \{\delta_a\}_{a \in \mathsf{ACT}} \rangle$ iff $(\mathfrak{S}, R) \models \Phi$. Furthermore, when $\mathfrak{S}$ is a regular PTS and R is a regular relation, we can compute in polynomial time a first-order formula Φ' over $\mathfrak{U}$ such that R is a probabilistic bisimulation over $\mathfrak{S}$ iff $\mathfrak{U} \models \Phi'$.*

This theorem implies the following result:

Theorem 2. *Given a regular relation $E \subseteq \Sigma^* \times \Sigma^*$ and a bounded-branching regular PTS $\mathfrak{S} = \langle S; \{\delta_a\}_{a \in \mathsf{ACT}} \rangle$, there exists an algorithm that either finds $(u, v) \in E$ which are not probabilistically bisimilar or finds a regular probabilistic bisimulation relation R over $\mathfrak{S}$ such that $E \subseteq R$ if one exists. The algorithm does not terminate iff E is contained in some probabilistic bisimulation relation but every probabilistic bisimulation R containing E is not regular.*

Note that when verifying parameterized systems we are typically interested in checking bisimilarity over *a set of pairs* (instead of just one pair) of configurations, and hence E in the above statement.

Proof of Theorem 2. To prove this, we provide two semi-algorithms, one for checking the existence of R and the other for showing that a pair $(v, w) \in E$ is a witness for non-bisimilarity.

By Theorem 1, we can enumerate all possible candidate regular relation R and effectively check that R is a probabilistic bisimulation over $\mathfrak{S}$. The condition that $E \subseteq R$ is a first-order property, and so can be checked effectively.

To show non-bisimilarity is recursively enumerable, observe that if we fix $(v, w) \in E$ and a number d, then the restrictions $\mathfrak{S}_v$ and $\mathfrak{S}_w$ to configurations that are of distance

at most d away from v and w (respectively) are finite PTS. Therefore, we can devise a semi-algorithm which enumerates all $(v, w) \in E$, and all probabilistic modal logic (PML) formulas [34] F over ACT containing only rational numbers (i.e. a formula of the form $\langle a \rangle_\mu F'$, where $\mu \in [0, 1]$ is a rational number, which is sufficient because we assume only rational numbers in the PTS). We need to check that $\mathfrak{S}_v, v \models F$, but $\mathfrak{S}_w, w \not\models F$. Model checking PML formulas over finite systems is decidable (in fact, the logic is subsumed by Probabilistic CTL [7]), which makes our check effective. □

4.3 Proof of Theorem 1

In the rest of the section, we shall give a proof of Theorem 1. Given a binary relation $R \subseteq S \times S$, we can write a first-order formula $F_{eq}(R)$ for checking that R is an equivalence relation:

$$\forall s, t, u \in S. R(s, s) \wedge (R(s, t) \Rightarrow R(t, s)) \wedge ((R(s, t) \wedge R(t, u) \Rightarrow R(s, u)).$$

We shall next define a formula $\varphi_a(p, q)$ for each $a \in \mathsf{ACT}$, such that R is a probabilistic bisimulation for $\mathfrak{S} = \langle S; \{\delta_a\}_{a \in \mathsf{ACT}} \rangle$ iff $(\mathfrak{S}, R) \models \Phi(R)$, where

$$\Phi(R) := F_{eq}(R) \wedge \forall p, q \in S.\ R(p, q) \Rightarrow \bigwedge\nolimits_{a \in \mathsf{ACT}} (\psi_a(p) \wedge \psi_a(q)) \vee \varphi_a(p, q). \quad (3)$$

The formula $\psi_a(s) := \forall s' \in S.\ \delta_a(s, s') = 0$ states that configuration s cannot move to any configuration through action a.

Before we describe $\varphi_a(p, q)$, we provide some intuition and define some intermediate macros. Fix configurations p and q. Informally, $\varphi_a(p, q)$ will first guess a set of configurations $u_1, \ldots, u_N$ containing the successors of p on action a, and a set of configurations $v_1, \ldots, v_N$ containing the successors of q on action a. Second, it will guess labellings $\alpha_1, \ldots, \alpha_N$ and $\beta_1, \ldots, \beta_N$ which correspond to partitionings of the configurations $u_1, \ldots, u_N$ and $v_1, \ldots, v_N$, respectively. The intuition is that the α's and β's "name" the partitions: if $\alpha_i = \alpha_j$ (resp. $\beta_i = \beta_j$), then u_i and u_j (resp. v_i and v_j) are guessed to be in the same partition. The formula then checks that the guessed partitioning is compatible with the equivalence relation R (i.e. if the labelling claims u_i and u_j are in the same partition, then indeed $R(u_i, u_j)$ holds), and that the probability masses of the partitions assigned by configurations p and q satisfy the constraint given in (1).

For the first part, we define a formula

$$\mathsf{succ}_a(w; u_1, \ldots, u_N) := \left(\bigwedge\nolimits_{1 \le i < j \le N} u_i \neq u_j\right) \wedge \left(\forall u \in S.\ \delta_a(w, u) \neq 0 \Rightarrow \bigvee\nolimits_{1 \le i \le N} u = u_i\right),$$

stating that the successors of configuration w on action a are among the N distinct configurations $u_1, \ldots, u_N$. Note that a configuration may have fewer than N successors. In this case, we can set the rest of the variables to arbitrary distinct configurations.

For the second part, we shall check that R is compatible with the guessed partitions, and that configurations p and q assign the same probability mass to the same partition.

Let $k_1, \ldots, k_n$ be a labelling for configurations $s_1, \ldots, s_n$. To check that the partitioning induced by the labelling is compatible with R, we need to express the condition that $k_i = k_j$ if and only if $R(s_i, s_j)$ holds. To this end, we define a formula

$$\mathsf{compat}_R(s_1, \ldots, s_n; k_1, \ldots, k_n) := \bigwedge\nolimits_{1 \le i < j \le n} (R(s_i, s_j) \Leftrightarrow k_i = k_j)\,.$$

Now, we are ready to define $\varphi_a(p, q)$:

$$\begin{aligned}\varphi_a(p,q) := \exists u_1, \ldots, u_N, v_1, \ldots, v_N \in S.\ \exists \alpha_1, \ldots, \alpha_N, \beta_1, \ldots, \beta_N \in \mathbb{N}.\\ \mathsf{succ}_a(p; u_1, \ldots, u_N) \wedge \mathsf{succ}_a(q; v_1, \ldots, v_N) \wedge \\ \mathsf{compat}_R(u_1, \ldots, u_N, v_1, \ldots, v_N; \alpha_1, \ldots, \alpha_N, \beta_1, \ldots, \beta_N) \wedge \\ \forall k \in \mathbb{N}.\ \Big(\sum\nolimits_{i:\ \alpha_i = k} \delta_a(p, u_i) = \sum\nolimits_{i:\ \beta_i = k} \delta_a(q, v_i)\Big)\,.\end{aligned} \tag{4}$$

With this definition, $\varphi_a(p, q)$ holds if and only if $p \xrightarrow{\rho}_a S' \Leftrightarrow q \xrightarrow{\rho}_a S'$ holds for any $\rho \ge 0$ and equivalence class $S' \in S/R$.

Example 3. Consider the PTS from Example 2. The configurations pXZ and rX are probabilistic bisimilar. This can be seen using a probabilistic bisimulation relation with equivalence classes $\{pX^kZ\} \cup \{rw : w \in \{X, X'\}^k\}$ for all $k \ge 0$ and $\{qX^{k+1}Z\} \cup \{rYw : w \in \{X, X'\}^k\}$ for all $k \ge 1$. The probabilistic bisimulation relation is definable as the symmetric closure of a regular relation R, where $(w_1, w_2) \in R$ iff

$$\begin{aligned}&(w_1 = w_2) \vee \\ &(w_1 \in pX^*Z \wedge w_2 \in r(X + X')^*\bot \wedge |w_1| = |w_2|) \vee \\ &(w_1 \in r(X + X')^* \wedge w_2 \in r(X + X')^* \wedge |w_1| = |w_2|) \vee \\ &(w_1 \in qX^*Z \wedge w_2 \in rY(X + X')^*\bot \wedge |w_1| = |w_2|) \vee \\ &(w_1 \in rY(X + X')^* \wedge w_2 \in rY(X + X')^* \wedge |w_1| = |w_2|).\end{aligned}$$

For this example, the formula (3) simplifies to $F_{eq}(R) \wedge \forall s, t \in S.\ \varphi_a(p, q)$ for the unique action a. This formula defines a condition that checks the bisimulation relation for all states symbolically. To see the formula in action, fix configurations pXZ and rX which are probabilistic bisimilar. In the PTS, pXZ has two successors, $qXXZ$ and pZ, each with probability 0.5, and rX has three successors, rYX with probability 0.3, rYX' with probability 0.2, and r with probability 0.5. In the formula for $\varphi_a(p, q)$, we can set the successors u_i of pXZ and the successors v_j of rX as above (the third "successor" u_3 is set to an arbitrary configuration not reachable from pXZ), and set $\alpha_1 = 1$, $\alpha_2 = 2$, $\beta_1 = \beta_2 = 1$, and $\beta_3 = 2$, corresponding to the equivalence classes of the bisimulation relation. One can check that the probability masses to these classes are the same.

We remark that the first-order theory of $\mathfrak{U}$ is sufficient to encode any probabilistic pushdown automaton, not just this example. □

We proceed to show that if R and δ_a are first-order definable over $\mathfrak{U}$ then so are ψ_a and φ_a. Suppose that δ_a is encoded using the ternary relation $\delta_a(x, y, z)$, as stated in the previous section. (We shall re-use the symbol δ here to avoid a clash of names).

We define $\psi_a(s) := \forall s' \in S.\, \forall z \in \mathbb{N}.\, \delta_a(s, s', z) \Leftrightarrow z = 0$. To define φ_a, the key point is to express the sum of transition probabilities in the logic. We use the fact that addition of integers in binary encoding is regular (see e.g. [9]), and write a formula that performs iterated addition. Formally, for each $a \in \mathsf{ACT}$ we define a formula χ_a such that

$$\chi_a(u; u_1, \ldots, u_N; \alpha_1, \ldots, \alpha_N; k; z) := \\ \exists z_1, \ldots, z_{N+1} \in \mathbb{N}.\, z_1 = 0 \wedge z_{N+1} = z \wedge \bigwedge\nolimits_{1 \le i \le N} \chi'_a(u, u_i, \alpha_i, k, z_i, z_{i+1}),$$

where

$$\chi'_a(u, u', \kappa, k, x, y) := (\kappa = k \wedge \exists z.\, \delta_a(u, u', z) \wedge y = x + z) \vee (\kappa \neq k \wedge y = x)$$

performs a single addition—we use the fact that addition "$y = x + z$" in binary is encodable as a regular relation—and $z_1, \ldots, z_{N+1}$ store the intermediate sums. Hence, given $k \in \mathbb{N}$, $u_1, \ldots, u_N, v_1, \ldots, v_N \in S$, and $\alpha_1, \ldots, \alpha_N, \beta_1, \ldots, \beta_N \in \mathbb{N}$,

$$\sum\nolimits_{i:\ \alpha_i = k} \delta_a(p, u_i) = \sum\nolimits_{i:\ \beta_i = k} \delta_a(q, v_i)$$

if and only if

$$\exists z \in \mathbb{N}.\, \chi_a(p; u_1, \ldots, u_N; \alpha_1, \ldots, \alpha_N; k; z) \wedge \chi_a(q; v_1, \ldots, v_N; \beta_1, \ldots, \beta_N; k; z).$$

It follows that $\varphi_a(p, q)$ defined in (4) can be encoded in the first-order theory of $\mathfrak{U}$.

Remark. Note that checking the validity of a given presentation of a regular PTS is algorithmic. To see this, suppose we are given a set of formulae $\{\delta_a(x, y, z)\}_{a \in \mathsf{ACT}}$ that is claimed to encode the probabilistic transition functions of a PTS with a branching bound N. Fix a formula δ_a. First, we need to check that for all $x \in S$, there are at most N distinct y's such that $\delta_a(x, y, z)$ satisfies $z \neq 0$. Second, we need to check that $[\![\delta_a]\!]$ is a function, i.e., $\forall x, y.\, \exists! z.\, \delta_a(x, y, z)$, where $\exists! z.\, \varphi(\bar{x}, z)$ is a shorthand for the formula asserting there exists precisely one z such that $\varphi(\bar{x}, z)$ is true. Third, we need to check that $[\![\delta_a]\!]$ encodes a mapping $S \to \{\bar{0}\} \cup \mathcal{D}_S$. The first two requirements are easily seen to be expressible as a first-order formula and hence is algorithmic over $\mathfrak{U}$. The third requirement amounts to checking the assertion that there exists $w_a \in \mathbb{N}$ satisfying

$$\forall x \in S.\, (\forall y \in S.\, \forall z \in \mathbb{N}.\, \delta_a(x, y, z) \Leftrightarrow z = 0) \vee \\ (\exists y_1, \ldots, y_N \in S.\, \exists z_1, \ldots, z_N \in \mathbb{N}. \\ \mathsf{succ}_a(x; y_1, \ldots, y_N) \wedge \bigwedge\nolimits_{1 \le i \le N} \delta_a(x, y_i, z_i) \wedge \sum\nolimits_{1 \le i \le N} z_i = w_a),$$

which is a first-order formula and is algorithmic over $\mathfrak{U}$ by the fact that summation of a fixed number of weights is regular (as shown earlier in this section). Finally, since all of the w_a's are expected to be the same common multiple of the denominators of the transition probabilities, we need to check that there is $w \in \mathbb{N}$ such that $w_a = w$ for all $a \in \mathsf{ACT}$. This is again algorithmic as we can pinpoint the exact value of each w_a by enumeration.

5 Application to Anonymity Verification

In this section, we show how to verify the anonymity property of cryptographic protocols via computation of probabilistic bisimulations. We shall first formalize the connection between the concepts of anonymity and probabilistic bisimulation. We then introduce a verification framework and apply it to verify the anonymity property of the dining cryptographers protocol [16] and the grades protocol [29].

A *(discrete time) Markov chain* (a.k.a. *DTMC*) is a structure $\mathfrak{M} := \langle S; \delta; L \rangle$ where S is a set of configurations, $\delta : S \to \mathcal{D}_S$ is a family of probability distributions, and $L : S \to \mathsf{ACT}$ is a labelling of the states. We shall use $\delta(s, s')$ to denote $\delta(s)(s')$, the transition probability from s to s'. A sequence $s_0 \dots s_n \in S^*$ is called a *path* of $\mathfrak{M}$ if $\delta(s_i, s_{i+1}) > 0$ for $i \in \{0, \dots, n-1\}$. The probability distribution induced by the paths in a DTMC can be defined using a standard cylinder construction (see e.g. [33]) as follows. Given a finite path $\pi := s_0 \cdots s_n \in S^*$, we set Run_π to be a *basic cylinder*, which is the set of all finite/infinite paths with π as a prefix. We associate this cylinder with probability $\Pr^{s_0}(Run_\pi) = \prod_{i=0}^{n-1} \delta(s_i, s_{i+1})$. This gives rise to a unique probability measure for the σ-algebra over the set of all paths from s_0.

Given a PTS $\mathfrak{S} := \langle S; \{\delta_a\}_{a \in \mathsf{ACT}} \rangle$, an *adversary* $f : S^* \to \mathsf{ACT}$ resolves the non-determinacy of $\mathfrak{S}$ and induces a DTMC $\mathfrak{S}_f := \langle S'; \delta'; L' \rangle$. Here $S' := S^* \cup \{\$\}$ contains all finite paths of $\mathfrak{S}$ plus a "sink state" $\$$ such that $\delta'(\pi) := I_{\$}$[2] if and only if either $\pi = \$$, or $\delta_{f(\pi)}$ is the zero function. We define $\delta'(\pi) := \delta_{f(\pi)}$ otherwise. The labelling of $\mathfrak{S}_f$ is defined as $L'(\$) := \bot$ and $L'(\pi) := f(\pi)$ for $\pi \in S^*$.

Given a DTMC $\langle S; \delta; L \rangle$, the *trace* of a path $\pi := s_0 \cdots s_n \in S^*$ is defined as $\tau(\pi) := L(s_0) \cdots L(s_n)$. A *trace event* $\mathcal{T}$ is a set of finite traces; the probability of $\mathcal{T}$ with respect to a configuration s is specified with $\Pr^s(\mathcal{T}) := \Pr^s(\bigcup \{Run_\pi : \tau(\pi) \in \mathcal{T}, \pi \text{ starts from } s\})$.

Now we are ready to define the concept of anonymity. Fix $\mathfrak{S} := \langle S; \{\delta_a\}_{a \in \mathsf{ACT}} \rangle$ and a set $\mathcal{I} \subseteq S$ of initial configurations. We say $\mathfrak{S}$ is *anonymous to an adversary* f if for all $s \in \mathcal{I}$ and trace event $\mathcal{T}$, the value of $\Pr^s(\mathcal{T})$ in $\mathfrak{S}_f$ is solely determined by $\mathcal{T}$. Intuitively, this means that the adversary cannot obtain any information about a specific initial configuration by experimenting on the system and observing the traces.

We shall only consider external adversaries in this paper. An adversary $f : S^* \to \mathsf{ACT}$ is *external* if $f(s_0 \cdots s_n) = f(s'_0 \cdots s'_n)$ when $L(s_i) = L(s'_i)$ for $i \in \{0, \dots, n\}$. That is, an external adversary takes action solely based on the trace she has observed so far. We call a PTS *anonymous* if it is anonymous to any external adversary. The following result establishes a connection between the anonymity property and probabilistic bisimulations.

Proposition 2. *Let $\mathfrak{S} := \langle S; \{\delta_a\}_{a \in \mathsf{ACT}} \rangle$ be a PTS and f be an external adversary for $\mathfrak{S}$. Then for all $u, v \in S$ such that $u \sim v$, $\Pr^u(\mathcal{T}) = \Pr^v(\mathcal{T})$ holds for any trace event $\mathcal{T}$ in $\mathfrak{S}_f$. That is, configurations u and v induce the same trace distribution in $\mathfrak{S}_f$.*

Based on Proposition 2, we propose a framework to verify the anonymity property of $\mathfrak{S}$ as follows. We first specify a "reference system" $\mathfrak{S}' := \langle S; \{\delta'_a\}_{a \in \mathsf{ACT}} \rangle$ that has

[2] Recall that I_s denotes the point distribution at s, namely $I_s(s) = 1$.

the same initial configurations and actions as those of $\mathfrak{S}$, except that the trace distribution of $\mathfrak{S}'_f$ is independent of specific initial configurations for any adversary f. We then try to find a bisimulation relation R between $\mathfrak{S}$ and the reference system $\mathfrak{S}'$ satisfying $R \supseteq \{(s, s') \in \mathcal{I} \times \mathcal{I}' : s = s'\}$. When such a relation R is found, we can conclude that the trace distribution of $\mathfrak{S}_f$ is also independent of the initial configurations for any adversary f, and hence prove the anonymity property of $\mathfrak{S}$.

The Dining Cryptographers Protocol. Dining cryptographers protocol [16] is a multi-party computation algorithm aiming to securely compute the XOR of the secret bits held by the participants. More precisely, consider a ring of $n \geq 3$ participants $p_0, \ldots, p_{n-1}$ such that each participant p_i holds a secret bit x_i. To compute $x_0 \oplus \cdots \oplus x_{n-1}$ without revealing information about the values of $x_0, \ldots, x_{n-1}$, the participants carry out a two-stage computation as follows: (i) Each two adjacent participants p_i, p_{i+1} compute a random bit b_i that is accessible only to them; (ii) Each participant p_i announces the value $a_i := x_i \oplus b_i \oplus b_{i-1}$[3] to the other participants. Hence, every participant p_i can observe the values of x_i, b_i, b_{i-1} and $a_0, \ldots, a_{n-1}$. It turns out that $a_0 \oplus \cdots \oplus a_{n-1} = x_0 \oplus \cdots \oplus x_{n-1}$, so all participants are able to compute the XOR of the secret bits after executing the protocol. Furthermore, the anonymity property of the protocol assures that any individual participant p_i cannot infer the values of the other secret bits from the information she has observed during the execution of the protocol.

We model the protocol as a length-preserving regular PTS. The configurations of a ring of n participants are encoded as words of size n. The initial configurations are words $w \in \{0, 1\}^*$ such that $w[i]$ represents x_i for $i \in \{0, \ldots, |w| - 1\}$. The transition relation consists of six transitions: observer non-deterministically tossing head (via action head), observer non-deterministically tossing tail (via action tail), non-observer tossing head with probability 0.5 (via action toss), non-observer tossing tail with probability 0.5 (via action toss), participant announcing zero (via action zero), and participant announcing one (via action one). The outcomes of the tosses by the observer are visible (i.e. as actions head and tail), while the outcomes of the tosses by the other participants are hidden (i.e. as action toss). Each maximal trace from an initial configuration of size n consists of n successive tossing actions, followed by n successive announcing actions. Starting from an initial configuration w and for $i \in \{0, \ldots, n-1\}$, the i-th toss action updates the value of $w[j]$ to $w[j] \oplus b_i$ for $j \in \{i, i+1\}$, where $b_i = 1$ if a head is tossed and $b_i = 0$ otherwise. Any configuration v reached after n tosses would satisfy $v[i] = x_i \oplus b_i \oplus b_{i-1}$ for $i \in \{0, \ldots, n-1\}$. The PTS then "prints out" the configuration by going through n announcement transitions via actions $a_0, \ldots, a_{n-1}$, such that a_i is one if $v[i] = 1$ and a_i is zero if $v[i] = 0$.

We consider the case where the first participant of the protocol is the observer. The maximal traces of the PTS in this case are in form of $t \cdot t'$, where $|t| = |t'|$, $t \in \{\mathsf{head}, \mathsf{tail}\}\,\mathsf{toss}^*\{\mathsf{head}, \mathsf{tail}\}$, and $t' \in \{\mathsf{zero}, \mathsf{one}\}^*$. For example, $\mathsf{head\ toss\ tail\ one\ zero\ zero}$ is a maximal trace starting from initial configuration 010. To prove anonymity, we define a reference system such that the initial configurations and the actions are the same as those of the original PTS, except that the announcements $a_0, \ldots, a_{n-1}$

[3] All arithmetical operations on the subscripts are performed modulo n to take the ring structure into account.

encoded in the maximal traces from an initial configuration w are uniformly distributed over $\{(a_0, \ldots, a_{n-1}) : a_0 \oplus \cdots \oplus a_{n-1} = w[0] \oplus \cdots \oplus w[n-1],\ a_0 = w[0] \oplus b_0 \oplus b_{n-1}\}$.[4] In this way, the distribution of the announcements is independent of the initial configuration once the values of $x_0 \oplus \cdots \oplus x_{n-1}$, x_0, b_0, and b_{n-1} (i.e. the information revealed to the first participant) are fixed. We then compute a probabilistic bisimulation between the original system and the reference system, establishing the anonymity property that the first participant cannot infer the secret bits of the other participants from the information she observes.

A generalized Dining Cryptographers Protocol. We have also considered a generalized dining cryptographers protocol where the secret messages $x_0, \ldots, x_{n-1}$ of the n participants are bit-vectors of the same size. Note that the set of the initial configurations is not regular when the size of the secret messages is parameterized. To construct a regular model, we allow a configuration to encode secret messages of different sizes, and devise the transition system such that an initial configuration w can finish the protocol (i.e. can have a trace containing all of the announcements $a_0, \ldots, a_{n-1}$) if and only if the messages encoded in w have same size. The resulting PTS is a regular system; it over-approximates the PTS of the generalized dining cryptographers protocol in the sense that the anonymity property of the former implies that of the latter.

The Grades Protocol. The grades protocol [29] is a multi-party computation algorithm aiming to securely compute the sum of the secrets held by the participants. The setting of the protocol is pretty similar to that of the dining cryptographers: given $n \geq 3$ and $g \geq 2$, we have a ring of n participants $p_0, \ldots, p_{n-1}$ where each participant p_i holds a secret $x_i \in \{0, \ldots, g-1\}$. Note that both g and n are parameterized in this protocol. The goal of the participants is to compute the sum $x_0 + \cdots + x_{n-1}$ without revealing information about the individual secrets. Define $M := (g-1) \cdot n + 1$. The protocol consists of two steps: (i) Each two adjacent participants p_i, p_{i+1} compute a random number $y_i \in \{0, \ldots, M-1\}$; (ii) Each participant p_i announces $a_i := (x_i + y_i - y_{i-1}) \bmod M$ to the other participants. After executing the protocol, the participants compute $a := a_0 + \cdots + a_{n-1} \bmod M$. Because of the ring structure, the y_i's will be cancelled out in the sum. Thus the value of a will equal to the sum of all secrets. The anonymity property of the protocol asserts that no participant can infer the secrets held by the other participants from the information she has observed.

We consider a variant of the grades protocol where M can be any power of two greater than $(g-1) \cdot n$. Observe that the same anonymity and correctness property of the original protocol also holds for this variant. To verify the anonymity property, we model an over-approximation of the protocol where the secrets are allowed to range over $\{0, \ldots, M-1\}$. This model is similar to the one we have constructed for the generalized dining cryptographers protocol except that, e.g., the XOR operations are now replaced with bitwise additions and negations. A reference system is specified such that the announcements $a_1, \ldots, a_{n-1}$ observed by the first participant p_0 are uniformly distributed over the values satisfying $a_0 + \cdots + a_{n-1} \bmod M = x_0 + \cdots + x_{n-1} \bmod M$.

[4] Such a distribution can be obtained by (i) choose $a_1, \ldots, a_{n-2} \in \{0, 1\}$ uniformly at random; (ii) set $a_0 = w[0] \oplus b_0 \oplus b_{n-1}$; (iii) set $a_{n-1} = a_0 \oplus \cdots \oplus a_{n-2} \oplus w[0] \oplus \cdots \oplus w[n-1]$.

Algorithm 1. Equivalence check for L^*

Input: Candidate automaton $\mathcal{H}$ over $\Sigma \times \Sigma$, PTS $\mathfrak{S}$, and relation $E \subseteq (\Sigma \times \Sigma)^*$.
Result: $\mathit{NoSolution}(v, w)$ if there is no bisimulation R with $E \subseteq R$.
$\mathit{PositiveCEX}(v, w)$ if $\mathcal{H}$ should accept (v, w), but does not;
$\mathit{NegativeCEX}(v, w)$ if $\mathcal{H}$ accepts (v, w), but should not;
$\mathit{Correct}$ if $\mathcal{H}$ is a correct bisimulation for PTS $\mathfrak{S}$ and $E \subseteq \mathcal{L}(\mathcal{H})$;

Check whether $E \subseteq \mathcal{L}(\mathcal{H})$, and whether $\mathfrak{S} \models \Phi(\mathcal{L}(\mathcal{H}))$ using the Φ from (3);
if *there is a counterexample of minimal length* n **then**
  Compute the greatest bisimulation $\bar{R}_n$ restricted to configurations of length n;
  if *there is* $(v \otimes w) \in E \setminus \bar{R}_n$ *with* $|v| = |w| = n$ **then**
    Output $\mathit{NoSolution}(v, w)$ and abort;
  else if *there is* $(v \otimes w) \in \mathcal{L}(\mathcal{H}) \setminus \bar{R}_n$ *with* $|v| = |w| = n$ **then**
    return $\mathit{NegativeCEX}(v, w)$;
  else if *there is* $(v \otimes w) \in \bar{R}_n \setminus \mathcal{L}(\mathcal{H})$ **then**
    return $\mathit{PositiveCEX}(v, w)$;
else
  return $\mathit{Correct}$;

By computing a probabilistic bisimulation between the original system and the reference system, we establish the anonymity property that the grades protocol is anonymous whenever M is chosen as a power of two with $M \geq (g-1) \cdot n + 1$.

6 Learning Probabilistic Bisimulations

We propose an automata learning method to automatically compute regular probabilistic bisimulations R, focusing on the case of *length-preserving* PTSs, which covers all examples given in the previous section. The approach uses active automata learning, for instance Angluin's L^* method [5] or refinements of it, to compute R. This approach is inspired by previous work on using active automata learning for invariant inference [18,54]. Our procedure assumes (i) as input a bounded-branching PTS $\mathfrak{S} = \langle S; \{\delta_a\}_{a \in \mathsf{ACT}} \rangle$, as well as a length-preserving regular relation $E \subseteq (\Sigma \times \Sigma)^*$ supposed to be covered by R; (ii) an effective way to check the correctness of R, i.e., a decision procedure in the sense of Theorem 1; and (iii) a procedure to compute the greatest probabilistic bisimulation $\bar{R}_n \subseteq (\Sigma \times \Sigma)^n$ for $\mathfrak{S}$ restricted to configurations of any length $n \in \mathbb{N}$. The last assumption can easily be satisfied for length-preserving PTSs. Indeed, such systems, restricted to configurations of length n, are finite-state, so that efficient existing methods [6,17,20,52] apply. A solution R is presented as a deterministic letter-to-letter transducer, i.e., as a deterministic finite-state automaton over the alphabet $\Sigma \times \Sigma$.

Since L^*-style learning requires the taught language to be uniquely defined, our approach attempts to learn a representation of the greatest *length-preserving* probabilistic bisimulation relation $\bar{R} \subseteq (\Sigma \times \Sigma)^*$, which is the unique bisimulation relation formed by the union of all length-preserving probabilistic bisimulations of $\mathfrak{S}$, i.e., $\bar{R} = \bigcup_{n \geq 1} \bar{R}_n$. Because $\bar{R}$ is not in general computable, the learning process might

diverge and fail to produce any probabilistic bisimulation. It can also happen that learning terminates, but yields a probabilistic bisimulation relation strictly smaller than $\bar{R}$.

The L^* method requires a teacher that is able to answer two kinds of queries:

- **membership queries**, i.e., whether a pair (v, w) of words should be accepted by the automaton to be learned. Since our learner tries to learn the greatest bisimulation, the teacher can answer this query by checking whether the configurations v, w are bisimilar; this is done by computing the greatest bisimulation $\bar{R}_{|v|}$ restricted to configurations of any length $|v| = |w|$, and checking whether or not $(v, w) \in \bar{R}_{|v|}$.
- **equivalence queries**, i.e., whether a candidate automaton $\mathcal{H}$ is the correct language to be learned. Such queries can essentially be answered by checking whether the language $\mathcal{L}(\mathcal{H})$ satisfies the formula $\Phi(R)$ from (3). The complete algorithm for answering equivalence queries is given in Algorithm 1. The algorithm first attempts to find a shortest counterexample to the proof rule. If a counterexample of length n is found, then the difference set $\mathcal{L}(\mathcal{H}) \mathbin{\Delta} \bar{R}_n$ must contain at least one pair of length n. Any of such pairs is a valid counterexample for automata learning since the learner tries to learn the greatest bisimulation. The teacher thus reports one such pair to be a positive or negative counterexample according to its membership in $\bar{R}_n$.

Properties of the Learning Algorithm. The learning procedure terminates when the teacher outputs *NoSolution* or returns *Correct* for an equivalence query. In the former case, the teacher explicitly provides a pair of non-bisimilar configurations in E. In the latter case, the procedure computes an automaton $\mathcal{H}$ such that $E \subseteq \mathcal{L}(\mathcal{H})$ and $\mathcal{L}(\mathcal{H})$ is a correct probabilistic bisimulation (as it satisfies the proof rule based on Theorem 1), though not necessarily the greatest one. Since all counterexamples reported by the teacher are contained in $\mathcal{L}(\mathcal{H}) \mathbin{\Delta} \bar{R}$, the learning procedure is guaranteed to terminate for PTSs where the greatest probabilistic bisimulation $\bar{R}$ is regular.

Optimization with Inductive Invariants. There is a natural way to optimize the learning procedure by only considering a *regular* inductive invariant Inv such that Inv contains the set of reachable configurations and $E \subseteq Inv \times Inv$. The optimization is done by simply replacing the greatest finite-length bisimulations $\bar{R}_i$ in Algorithm 1, and when answering membership queries, with the greatest bisimulation $\bar{R}_i^I = \bar{R}_i \cap Inv$ on the inductive invariant. Since $\bar{R}_i^I$ can be a lot smaller than $\bar{R}_i$, this can lead to significant speed-ups. Note that a bisimulation R' on Inv can be extended to a bisimulation R on all configurations by setting $R = R' \cup \{(v, v) : v \notin Inv\}$. The inductive invariant Inv may be manually specified, or automatically generated using techniques like in [18,54].

Experimental Results and Conclusion. We have implemented a prototype in Scala to test our learning method. Given a PTS specified over $\mathfrak{U}$, our tool first translates it to WS1S formulas and obtains finite automata for these formulas using the Mona tool [30]. Our prototype then applies the L^* learning procedure as described in this section, including the optimization to consider only the configurations of valid format. When answering an equivalence query, our tool invokes Mona to verify candidate automata and obtain counterexamples (line 1–2 of Algorithm 1). We use the prototype tool to prove the anonymity property of the three protocols described in Sect. 5. The proofs

Table 1. Experimental results. For each case study, we list the size of the final proof produced by our tool, the time taken by Mona to verify the candidate automata, the time taken by our tool to compute the fixed-length bisimulations, and the total computation time of the learning procedure. Experiments are run on a Windows laptop with 2.4 GHz Intel i5 processor and 2 GB memory limit.

Case study	#states	#trans	Mona	Bisim	Total
Dining cryptographers, single-bit	13	832	2 s	2 s	6 s
Dining cryptographers, multi-bit	16	1024	3 s	24 s	28 s
The grades protocol	25	1600	5 s	28 s	35 s

generated by our tool are finite-state automata encoding the desired probabilistic bisimulation relations. The experimental results are summarized in Table 1.

References

1. Abdulla, P.A., Henda, N.B., Mayr, R.: Decisive Markov chains. Log. Methods Comput. Sci. **3**(4), 1–32 (2007)
2. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J., Saksena, M.: Regular model checking for LTL(MSO). STTT **14**(2), 223–241 (2012)
3. Abdulla, P.A., Jonsson, B., Nilsson, M., Saksena, M.: A survey of regular model checking. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 35–48. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28644-8_3
4. Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M. (eds.): Deductive Software Verification - The KeY Book - From Theory to Practice. LNCS, vol. 10001. Springer, Cham (2016)
5. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
6. Baier, C.: Polynomial time algorithms for testing probabilistic bisimulation and simulation. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 50–61. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61474-5_57
7. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)
8. Benedikt, M., Libkin, L., Schwentick, T., Segoufin, L.: Definable relations and first-order query languages over strings. J. ACM **50**(5), 694–751 (2003)
9. Blumensath, A.: Automatic structures. Diploma thesis, RWTH-Aachen (1999)
10. Blumensath, A., Grädel, E.: Finite presentations of infinite structures: automata and interpretations. Theory Comput. Syst. **37**(6), 641–674 (2004)
11. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large (extended abstract). In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 223–235. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_24
12. Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2013, Rome, Italy 23–25 January 2013, pp. 457–468 (2013)
13. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 403–418. Springer, Heidelberg (2000). https://doi.org/10.1007/10722167_31

14. Bradley, A.R., Manna, Z.: The Calculus of Computation: Decision Procedures with Applications to Verification. Springer, Heidelberg (1998)
15. Chatzikokolakis, K., Norman, G., Parker, D.: Bisimulation for demonic schedulers. In: de Alfaro, L. (ed.) FoSSaCS 2009. LNCS, vol. 5504, pp. 318–332. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00596-1_23
16. Chaum, D.: The dining cryptographers problem: unconditional sender and recipient untraceability. J. Cryptol. **1**(1), 65–75 (1988)
17. Chen, D., van Breugel, F., Worrell, J.: On the complexity of computing probabilistic bisimilarity. In: Birkedal, L. (ed.) FoSSaCS 2012. LNCS, vol. 7213, pp. 437–451. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28729-9_29
18. Chen, Y., Hong, C., Lin, A.W., Rümmer, P.: Learning to prove safety over parameterised concurrent systems. In: 2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, 2–6 October 2017, pp. 76–83 (2017)
19. Colcombet, T., Löding, C.: Transforming structures by set interpretations. Log. Methods Comput. Sci. **3**(2), 1–36 (2007)
20. Derisavi, S., Hermanns, H., Sanders, W.H.: Optimal state-space lumping in Markov chains. Inf. Process. Lett. **87**(6), 309–315 (2003)
21. Esparza, J., Etessami, K.: Verifying probabilistic procedural programs. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 16–31. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30538-5_2
22. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge (2003)
23. Fiedor, T., Holík, L., Janků, P., Lengál, O., Vojnar, T.: Lazy automata techniques for WS1S. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 407–425. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_24
24. Flanagan, C., Leino, K., Lillibridge, M., Nelson, G., Saxe, J.B., Stata, R.: Extended static checking for Java. In: PLDI 02: Programming Language Design and Implementation, pp. 234–245. ACM (2002)
25. Forejt, V., Jancar, P., Kiefer, S., Worrell, J.: Bisimilarity of probabilistic pushdown automata. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, Hyderabad, India, 15–17 December 2012, pp. 448–460 (2012)
26. Garg, P., Neider, D., Madhusudan, P., Roth, D.: Learning invariants using decision trees and implication counterexamples. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, 20–22 January 2016, pp. 499–512 (2016)
27. Habermehl, P., Vojnar, T.: Regular model checking using inference of regular languages. Electr. Notes Theor. Comput. Sci. **138**(3), 21–36 (2005)
28. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: a tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006). https://doi.org/10.1007/11691372_29
29. Kiefer, S., Murawski, A.S., Ouaknine, J., Wachter, B., Worrell, J.: APEX: an analyzer for open probabilistic programs. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 693–698. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_51
30. Klarlund, N., Møller, A.: Mona version 1.4: User manual. BRICS, Department of Computer Science, University of Aarhus Denmark (2001)
31. Klarlund, N., Møller, A., Schwartzbach, M.I.: MONA implementation secrets. Int. J. Found. Comput. Sci. **13**(4), 571–586 (2002). World Scientific Publishing Company. Earlier version in Proc. 5thInternational Conference on Implementation and Application of Automata (CIAA) 2000, Springer-Verlag LNCS vol. 2088

32. Kundu, S., Tatlock, Z., Lerner, S.: Proving optimizations correct using parameterized program equivalence. In: Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2009, pp. 327–337. ACM, New York (2009)
33. Kwiatkowska, M.Z., Model checking for probability and time: from theory to practice. In: 18th IEEE Symposium on Logic in Computer Science (LICS 2003), Ottawa, Canada, 22–25 June 2003, Proceedings, p. 351 (2003)
34. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Inf. Comput. **94**(1), 1–28 (1991)
35. Leino, K.R.M.: Dafny: an automatic program verifier for functional correctness. In: Clarke, E.M., Voronkov, A. (eds.) LPAR 2010. LNCS (LNAI), vol. 6355, pp. 348–370. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17511-4_20
36. Lengál, O., Lin, A.W., Majumdar, R., Rümmer, P.: Fair termination for parameterized probabilistic concurrent systems. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 499–517. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_29
37. Lin, A.W., Rümmer, P.: Liveness of randomised parameterised systems under arbitrary schedulers. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 112–133. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_7
38. Löding, C., Madhusudan, P., Neider, D.: Abstract learning frameworks for synthesis. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 167–185. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_10
39. Milner, R.: Communication and Concurrency. Prentice Hall, Upper Saddle River (1989)
40. Neider, D., Jansen, N.: Regular model checking using solver technologies and automata learning. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 16–31. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_2
41. Neider, D., Topcu, U.: An automaton learning approach to solving safety games over infinite graphs. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 204–221. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_12
42. Nilsson, M.: Regular model checking. Ph.D. thesis, Uppsala Universitet (2005)
43. Padon, O., Losa, G., Sagiv, M., Shoham, S.: Paxos made EPR: decidable reasoning about distributed protocols. In: PACMPL, 1(OOPSLA), pp. 108:1–108:31 (2017)
44. Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: safety verification by interactive generalization. In: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, 13–17 June 2016, pp. 614–630 (2016)
45. PRISM case study: Dining Cryptographers. http://www.prismmodelchecker.org/casestudies/diningcrypt.php
46. Rubin, S.: Automatic structures. Ph.D. thesis, University of Auckland, New Zealand (2004)
47. Sénizergues, G.: The bisimulation problem for equational graphs of finite out-degree. SIAM J. Comput. **34**(5), 1025–1106 (2005)
48. Srba, J.: Roadmap of Infinite Results. Formal Models and Semantics, vol. 2. World Scientific Publishing Co., Singapore (2004)
49. To, A.W.: Model checking infinite-state systems: generic and specific approaches. Ph.D. thesis, LFCS, School of Informatics, University of Edinburgh (2010)
50. To, A.W., Libkin, L.: Recurrent reachability analysis in regular model checking. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 198–213. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89439-1_15
51. To, A.W., Libkin, L.: Algorithmic metatheorems for decidable LTL model checking over infinite systems. In: Ong, L. (ed.) FoSSaCS 2010. LNCS, vol. 6014, pp. 221–236. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12032-9_16

52. Valmari, A., Franceschinis, G.: Simple $O(m \log n)$ time Markov chain lumping. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 38–52. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_4
53. Vardhan, A.: Learning to verify systems. Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (2006)
54. Vardhan, A., Sen, K., Viswanathan, M., Agha, G.: Learning to verify safety properties. In: Davies, J., Schulte, W., Barnett, M. (eds.) ICFEM 2004. LNCS, vol. 3308, pp. 274–289. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30482-1_26

Semi-quantitative Abstraction and Analysis of Chemical Reaction Networks

Milan Češka[1(✉)] and Jan Křetínský[2]

[1] Brno University of Technology, FIT,
IT4I Centre of Excellence, Brno, Czech Republic
ceskam@fit.vutbr.cz
[2] Technical University of Munich, Munich, Germany

Abstract. Analysis of large continuous-time stochastic systems is a computationally intensive task. In this work we focus on population models arising from chemical reaction networks (CRNs), which play a fundamental role in analysis and design of biochemical systems. Many relevant CRNs are particularly challenging for existing techniques due to complex dynamics including stochasticity, stiffness or multimodal population distributions. We propose a novel approach allowing not only to predict, but also to explain both the transient and steady-state behaviour. It focuses on qualitative description of the behaviour and aims at quantitative precision only in orders of magnitude. First we build a compact understandable model, which we then crudely analyse. As demonstrated on complex CRNs from literature, our approach reproduces the known results, but in contrast to the state-of-the-art methods, it runs with virtually no computational cost and thus offers unprecedented scalability.

1 Introduction

Chemical Reaction Networks (CRNs) are a versatile language widely used for *modelling and analysis* of biochemical systems [12] as well as for high-level *programming* of molecular devices [8,40]. They provide a compact formalism equivalent to Petri nets [37], Vector Addition Systems (VAS) [29] and distributed population protocols [3]. Motivated by numerous potential applications ranging from system biology to synthetic biology, various techniques allowing simulation and formal analysis of CRNs have been proposed [2,9,21,24,39], and embodied in the design process of biochemical systems [20,25,32]. The time-evolution of CRNs is governed by the Chemical Master Equation (CME), which describes the probability of the molecular counts of each chemical species. Many important biochemical systems lead to complex dynamics that includes *state space explosion, stochasticity, stiffness, and multimodality* of the population distributions

This work has been supported by the Czech Science Foundation grant No. GA19-24397S, the IT4Innovations excellence in science project No. LQ1602, and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification".

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 475–496, 2019.
https://doi.org/10.1007/978-3-030-25540-4_28

[23,44], and that fundamentally limits the class of systems the existing techniques can effectively handle. More importantly, biologist and engineers often seek for plausible explanations why the system under study has or has not the required behaviour. In many cases, a set of system simulations/trajectories or population distributions is not sufficient and the ability to provide an accurate explanation for the temporal or steady-state behaviour is another major challenge for the existing techniques.

In order to cope with the computational complexity of the analysis and in order to obtain explanations of the behaviour, we shift the focus from quantitatively precise results to a more qualitative analysis, closer to how a human would behold the system. Yet we insist on providing at least rough timing information on the behaviour as well as rough classification of probability of different behaviours at the extent of "very likely", "few percent", "barely possible", so that we can conclude on issues such as time to extinction or bimodality of behaviour. This gives rise to our *semi-quantitative* approach. We stipulate that analyses in this framework reflect quantities in orders of magnitude, both for time duration and probabilities, but not more than that. This paradigm shift is reflected on two levels: (1) We abstract systems into semi-quantitative models. (2) We analyse systems in a semi-quantitative way. While each of the two can be combined with a traditional abstraction/analysis, when combined together they provide powerful means to understand systems' behaviour with virtually no computational cost.

Semi-quantitative Models. The states of the models contain information on the current amount of objects of each species as an interval spanning often several orders of magnitude, unless instructed otherwise. For instance, if an amount of a certain species is to be closely monitored (as a part of the input specification/property of the system) then this abstraction can be finer. Similarly, whenever the analysis of a previous version of the abstraction points to the lack of precision in certain states, preventing us to conclude which of the possible behaviours is prevalent, the corresponding refinement can take place. Further, the rates of the transitions are also captured only with such imprecision. The crucial point allowing for existence of such models that are small, yet faithful, is our concept of *acceleration*. It captures certain *sequences* of transitions. It eliminates most of the non-determinism that paralyses other types of abstractions, which are too over-approximative, unable to conclude anything, but safety properties.

Semi-quantitative Analysis. Instead of performing exact transient or steady-state analysis, we can consider most probable transitions and then carefully lift this to most probable temporal behaviours. Technically, this is done by *alternating between transient and steady-state analysis* where only some rates and transitions are taken into account at different stages. In order to further facilitate the resulting insight of the human on the result of the analysis, we provide an algorithm to perform this analysis with virtually no computation effort and thus possibly manually. The trivial computations immediately pinpoint why certain

behaviours occur. Moreover, less likely behaviours can also be identified easily, to any desired degree of improbability (dozens of percent, promilles etc.).

To summarise, the first step yields tiny models, allowing for a synoptic observation of the model; due to their size these models can be either analysed easily using standard means, or can be subject to the second step. The second step provides an efficient approximative analysis, which is also very illustrative due to the limited use of quantities. It can be applied to any system; however, it is particularly interesting in connection with the models coming from the first step since (i) no extra effort (size, computation) is wasted on overly precise treatment that is ignored by the other step, and (ii) together they yield an understandable explanation of the behaviour. An entertaining feature of this paradigm is that the stiffer (with rates at hugely different time scales) the system is the easier it is to analyse.

To demonstrate the capabilities of our approach, we consider three challenging and biologically relevant case studies that have been used in literature to evaluate state-of-the-art methods for the CRN analysis. It has been shown that many approaches fail, either due to time-outs or incapability to capture differences in behaviours, and some tailored ones require considerable computational effort, e.g. an hour of computation. Our experiments clearly show that the proposed approach can deliver results that yield qualitatively same information, more understanding and can be computed in minutes by hand (or within a fraction of a second by computer).

Our contribution can be summarized as follows:

- We propose a novel *semi-quantitative* framework for analysis of CRN and similar population models, focusing on explainability of the results and low complexity, with quantitative precision limited to orders of magnitude.
- An algorithm for abstracting CRNs into semi-quantitative models based on interval abstraction of the species population and on transition acceleration.
- An algorithm for semi-quantitative analysis that replaces exact numerical computation by exploring the most probable transitions and alternating transient and steady-state analysis.
- We consider three challenging CRNs thoroughly studied in literature and demonstrate that the semi-quantitative abstraction and analysis gives us a unique tool that is able to accurately predict and explain both transient and steady-state behaviour of complex CRNs in a fraction of a second.

Related Work

To the best of our knowledge, there does not exist any abstraction of CRNs similar to the proposed approach. Indeed, there exist various abstraction and approximation schemes for CRNs that improve the performance and scalability of both the simulation-based and the numerical-based techniques. In the following paragraphs, we discuss the most relevant directions and the links to our approach.

Approximate Semantics for CRNs. For CRNs including large populations of species, fluid (mean-field) approximation techniques can be applied [5] and extended to approximate higher-order moments [15]: these deterministic approximations lead to a set of ordinary differential equations (ODEs). An alternative is to approximate the CME as a continuous-state stochastic process. The Linear Noise Approximation (LNA) is a Gaussian process which has been derived as an approximation of the CME [16,44] and describes the time evolution of expectation and variance of the species in terms of ODEs. Recently, an aggregation scheme over ODEs that aims at understanding the dynamics of large CRNs has been proposed in [10]. In contrast to our approach, the deterministic approximations cannot adequately capture the stochasticity of CRNs caused by low population species.

To mitigate this drawback, various *hybrid models* have been proposed. The common idea of these models is as follows: the dynamics of low population species is described by the discrete stochastic process and the dynamics of large population species is approximated by a continuous process. The particular hybrid models differ in the approximation of the large population species. In [27], a pure deterministic semantics for large population species is used. The moment-based description for medium/high-copy number species was used in [24]. The LNA approximation and an adaptive partitioning of the species according to leap conditions (that is more general than partitioning based on population thresholds) was proposed in [9]. All hybrid models have to deal with interactions between low and large population species. In particular, the dynamics of the stochastic process describing the low-population species is conditioned by the continuous-state describing the concentration of the large-population species. The numerical analysis of such conditioned stochastic process is typically a computationally demanding task that limits the scalability.

In contrast, our approach does not explicitly partition the species, but rather abstracts the concrete species population using an interval abstraction and tries to effectively capture both the stochastic and the deterministic behaviour with the help of the accelerated transitions. As we already emphasised, the proposed abstraction and analysis avoids any numerical computation of precise quantities.

Reduction Techniques for Stochastic Models. A widely studied reduction method for Markov models is state aggregation based on lumping [6] or (bi-)simulation equivalence [4], with the latter notion in its exact [33] or approximate [13] form. Approximate notions of equivalence have led to new abstraction/refinement techniques for the numerical verification of Markov models over finite [14] as well as uncountably-infinite state spaces [1,41,42]. Several approximate aggregation schemes leveraging the structural properties of CRNs were proposed [17,34,45]. Abate et al. proposed an adaptive aggregation that gives formal guarantees on the approximation error, but typically provide lower state space reductions [2]. Our approach shares the idea of abstracting the state space by aggregating some states together. Similarly to [17,34,45], we partition the state space based on the species population, i.e. we also introduce the population levels. In contrast to the aforementioned aggregation schemes, we propose a

novel abstraction of the transition relation based on the acceleration. It allows us to avoid the numerical solution of the approximate CME and thus achieve a better reduction while providing an accurate predication of the system behaviour.

Alternative methods to deal with large/infinite state spaces are based on a state truncation trying to eliminate insignificant states, i.e., states reached only with a negligible probability. These methods, including finite state projections [36], sliding window abstractions [26], or fast adaptive uniformisation [35], are able to quantify the total probability mass that is lost due to the truncation, but typically cannot effectively handle systems involving a stiff behaviour and multimodality [9].

Simulation-Based Analysis. Transient analysis of CRNs can be performed using the Stochastic Simulation Algorithm (SSA) [21]. Note that the SSA produces a single realisation of the stochastic process, whereas the stochastic solution of CME gives the probability distribution of each species over time. Although simulation-based analysis is generally faster than direct solution of the stochastic process underlying the given CRN, obtaining good accuracy necessitates potentially large numbers of simulations and can be very time consuming.

Various partitioning schemes for species and reactions have been proposed for the purpose of speeding up the SSA in multi-scale systems [23,38,39]. For instance, Yao et al. introduced the slow-scale SSA [7], where they distinguish between fast and slow species. Fast species are then treated assuming they reach equilibrium much faster than the slow ones. Adaptive partitioning of the species has been considered in [19,28]. In contrast to the simulation-based analysis, our approach (i) provides a compact explanation of the system behaviour in the form of tiny models allowing for a synoptic observation and (ii) can easily reveal less probable behaviours.

2 Chemical Reaction Networks

In this paper, we assume familiarity with standard verification of (continuous-time) probabilistic systems, e.g. [4]. For more detail, see [11, Appendix].

CRN Syntax. A *chemical reaction network (CRN)* $\mathcal{N} = (\Lambda, \mathcal{R})$ is a pair of finite sets, where Λ is a set of *species*, $|\Lambda|$ denotes its size, and $\mathcal{R}$ is a set of reactions. Species in Λ interact according to the reactions in $\mathcal{R}$. A *reaction* $\tau \in \mathcal{R}$ is a triple $\tau = (r_\tau, p_\tau, k_\tau)$, where $r_\tau \in \mathbb{N}^{|\Lambda|}$ is the *reactant complex*, $p_\tau \in \mathbb{N}^{|\Lambda|}$ is the *product complex* and $k_\tau \in \mathbb{R}_{>0}$ is the coefficient associated with the rate of the reaction. r_τ and p_τ represent the stoichiometry of reactants and products. Given a reaction $\tau_1 = ([1,1,0],[0,0,2],k_1)$, we often refer to it as $\tau_1 : \lambda_1 + \lambda_2 \xrightarrow{k_1} 2\lambda_3$.

CRN Semantics. Under the usual assumption of mass action kinetics, the *stochastic* semantics of a CRN $\mathcal{N}$ is generally given in terms of a discrete-state, continuous-time stochastic process $\mathbf{X}(\mathbf{t}) = (X_1(t), X_2(t), \ldots, X_{|\Lambda|}(t), t \geq 0)$ [16]. The *state change* associated to the reaction τ is defined by $\upsilon_\tau = p_\tau - r_\tau$, i.e. the state $\mathbf{X}$ is changed to $\mathbf{X}' = \mathbf{X} + \upsilon_\tau$, which we denote as $\mathbf{X} \xrightarrow{\tau} \mathbf{X}'$. For example,

for τ_1 as above, we have $\upsilon_{\tau_1} = [-1, -1, 2]$. For a reaction to happen in a state $\mathbf{X}$, all reactants have to be in sufficient numbers. The *reachable state space* of $\mathbf{X}(\mathbf{t})$, denoted as $\mathbf{S}$, is the set of all states reachable by a sequence of reactions from a given *initial state* $\mathbf{X}_0$. The set of reactions changing the state $\mathbf{X}_i$ to the state $\mathbf{X}_j$ is denoted as $\mathsf{reac}(\mathbf{X}_i, \mathbf{X}_j) = \{\tau \mid \mathbf{X}_i \xrightarrow{\tau} \mathbf{X}_j\}$.

The behaviour of the stochastic system $\mathbf{X}(\mathbf{t})$ can be described by the (possibly infinite) continuous-time Markov chain (CTMC) $\gamma(\mathcal{N}) = (\mathbf{S}, \mathbf{X}_0, \mathbf{R})$ where the transition matrix $\mathbf{R}(i, j)$ gives the probability of a transition from $\mathbf{X}_i$ to $\mathbf{X}_j$. Formally,

$$\mathbf{R}(i,j) = \sum_{\tau \in \mathsf{reac}(\mathbf{X}_i, \mathbf{X}_j)} k_\tau \cdot C_{\tau,i} \quad \text{where} \quad C_{\tau,i} = \prod_{\ell=1}^{N} \binom{\mathbf{X}_{i,\ell}}{r_\ell} \tag{R}$$

corresponds to the population dependent term of the *propensity function* where $\mathbf{X}_{i,\ell}$ is ℓth component of the state $\mathbf{X}_i$ and r_ℓ is the stoichiometric coefficient of the ℓ-th reactant in the reaction τ. The CTMC $\gamma(\mathcal{N})$ is the accurate representation of CRN $\mathcal{N}$, but—even when finite—not scalable in practice because of the state space explosion problem [25,31].

3 Semi-quantitative Abstraction

In this section, we describe our abstraction. We derive the desired CTMC conceptually in several steps, which we describe explicitly, although we implement the construction of the final system directly from the initial CRN.

3.1 Over-Approximation by Interval Abstraction and Acceleration

Given a CRN $\mathcal{N} = (\Lambda, \mathcal{R})$, we first consider an interval continuous-time Markov decision process (interval CTMDP[1]), which is a finite abstraction of the infinite $\gamma(\mathcal{N})$. Intuitively, abstract states are given by intervals on sizes of populations with an additional specific that the abstraction captures enabledness of reactions. The transition structure follows the ideas of the standard may abstraction and of the three-valued abstraction of continuous-time systems [30]. A technical difference in the latter point is that we abstract rates into intervals instead of uniformising the chain and then only abstracting transition probabilities into intervals; this is necessary in later stages of the process. The main difference is that we also treat certain sequences of actions, which we call acceleration.

Abstract Domains. The first step is to define the abstract domain for the population sizes. For every species $\lambda \in \Lambda$, we define a finite partitioning A_λ of $\mathbb{N}$ into intervals, reflecting the rough size of the population. Moreover, we want the abstraction to reflect whether a reaction is enabled. Hence we require that

[1] Interval CTMDP is a CTMDP with lower/upper bounds on rates. Since it serves only as an intermediate formalism to ease the presentation, we refrain from formalising it here.

$\{0\} \in A_\lambda$ for the case when the coefficients of this species as a reactant is always 0 or 1; in general, for every $i < \max_{\tau \in \mathcal{R}} r_\tau(\lambda)$ we require $\{i\} \in A_\lambda$.

The abstraction $\alpha_\lambda(n)$ of a number n of a species λ is then the $I \in A_\lambda$ for which $n \in I$. The state space of $\alpha(\mathcal{N})$ is the product $\prod_{\lambda \in \Lambda} A_\lambda$ of the abstract domains with the point-wise defined abstraction $\alpha(\boldsymbol{n})_\lambda = \alpha_\lambda(\boldsymbol{n}_\lambda)$.

The abstract domain for the rates according to (R) is the set of all real intervals.

Transitions from an abstract state are defined as the may abstraction as follows. Since our abstraction reflect enabledness, the same set of action is enabled in all concrete states of a given abstract state. The targets of the action in the abstract setting are abstractions of all possible concrete successors, i.e. $succ(s, a) := \{\alpha(\boldsymbol{n}) \mid \boldsymbol{m} \in s, \boldsymbol{m} \xrightarrow{a} \boldsymbol{n}\}$, in other words, the transitions enabled in at least one of the respective concrete states. The abstract rate is the smallest interval including all the concrete rates of the respective concrete transitions. This can be easily computed by the corner-points abstraction (evaluating only the extremum values for each species) since the stoichiometry of the rates is monotone in the population sizes.

High-Level of Non-determinism. The (more or less) standard style of the abstraction above has several drawbacks—mostly related to the high degree of non-determinism for rates—which we will subsequently discuss.

Firstly, in connection with the abstract population sizes, transitions to different sizes only happen non-deterministically, leaving us unable to determine which behaviour is probable. For example, consider the simple system given by $\lambda \xrightarrow{d} \emptyset$ with $k_d = 10^{-4}$ so the degradation happens on average each 10^4 seconds. Assume population discretisation into $[0], [1..5], [6..20], [21..\infty)$ with abstraction depicted in Fig. 1. While the original system obviously moves from $[6..20]$ to $[1..5]$ very probably in less than $15 \cdot 10^4$ seconds, the abstraction cannot even say that it happens, not to speak of estimating the time.

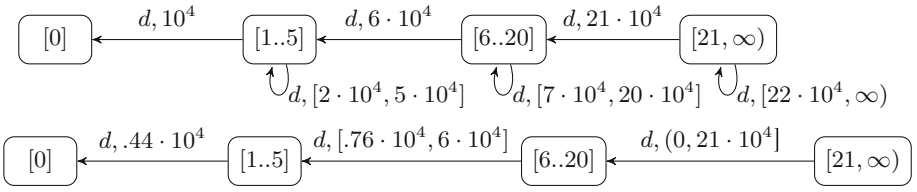


Fig. 1. Above: Interval CTMDP abstraction with intervals on rates and non-determinism. Below: Interval CTMC abstraction arising from acceleration.

Acceleration. To address this issue, we drop the non-deterministic self-loops and transitions to higher/lower populations in the abstract system.[2] Instead,

[2] One can also preserve the non-determinism for the special case when one of the transitions leads to a state where some action ceases to be enabled. While this adds more precision, the non-determinism in the abstraction makes it less convenient to handle.

we *"accelerate"* their effect: We consider sequences of these actions that in the concrete system have the effect of changing the population level. In our example above, we need to take the transition 1 to 13 times from [6..20] with various rates depending on the current concrete population, in order to get to [1..5]. This makes the precise timing more complicated to compute. Nevertheless, the expected time can be approximated easily: here it ranges from $\frac{1}{6} \cdot 10^4 = 0.17 \cdot 10^4$ (for population 6) to roughly $(\frac{1}{20} + \frac{1}{19} + \cdots + \frac{1}{6}) \cdot 10^4 = 1.3 \cdot 10^4$ (for population 20). This results in an interval CTMC.[3]

Concurrency in Acceleration. The accelerated transitions can due to higher number of occurrences be considered continuous or deterministic, as opposed to discrete stochastic changes as distinguished in the hybrid approach. The usual differential equation approach would also take into account other reactions that are modelled deterministically and would combine their effect into one equation. In order to simplify the exposition and computation and—as we see later—without much loss of precision, we can consider only the fastest change (or non-deterministically more of them if their rates are similar).[4]

3.2 Operational Semantics: Concretisation to a Representative

The next disadvantage of classical abstraction philosophy, manifested in the interval CTMC above is that the precise-valued intervals on rates imply high computational effort during the analysis. Although the system is smaller, standard transient analysis is still quite expensive.

Concretisation. In order to deal with this issue, the interval can be approximated roughly by the expected time it would take for an average population in the considered range, in our example the "average" representative is 13. Then the first transition occurs with rate $13 \cdot 10^{-4} = 10^{-3}$ and needs to happen 7 times, yielding expected time $7/13 \cdot 10^4 = 0.5 \cdot 10^4$ (ignoring even the precise slow downs in the rates as the population decreases). Already this very rough computation yields relative precision with factor 3 for all the populations in this interval, thus yielding the correct order of magnitude with virtually no effort. We lift the concretisation naturally to states and denote the concretisation of abstract state s by $\gamma(s)$. The complete procedure is depicted in Algorithm 1.

The concretisation is one of the main points where we deliberately drop a lot of quantitative information, while still preserving some to conclude on big quantitative differences. Of course, the precision improves with more precise abstract domains and also with higher differences on the original rates.

[3] The waiting times are not distributed according to the rates in the intervals. It is only the expected waiting time (reciprocal of the rate) that is preserved. Nevertheless, for ease of exposition, instead of labelling the transitions with expected waiting times we stick to the CTMC style with the reciprocals and formally treat it as if the label was a real rate.

[4] Typically the classical concurrency diamond appears and the effect of the other accelerated reactions happen just after the first one.

Algorithm 1. Semi-quantitative abstraction CTMC $\alpha(\mathcal{N})$

$A \leftarrow \prod_{\lambda \in \Lambda} A_\lambda$ ▷ States
for $\boldsymbol{a} \in A$ **do** ▷ Transitions
  $\boldsymbol{c} \leftarrow \gamma(\boldsymbol{a})$ ▷ Concrete representative
  for each τ enabled in $\boldsymbol{c}$ **do**
    $r \leftarrow$rate of τ in $\boldsymbol{c}$ ▷ According to (R)
    $\boldsymbol{a}' \leftarrow \alpha(\boldsymbol{c} + \upsilon_\tau)$ ▷ Successor
    set $\boldsymbol{a} \xrightarrow{\tau} \boldsymbol{a}'$ with rate r
  for self-loop $\boldsymbol{a} \xrightarrow{\tau} \boldsymbol{a}$ **do** ▷ Accelerate self-loops
    $n_\tau \leftarrow \min\{n \mid \alpha(\boldsymbol{c} + n \cdot \upsilon_\tau) \neq \boldsymbol{a}\}$ ▷ the number of τ to change the abstract state
    $\boldsymbol{a}' \leftarrow \alpha(\boldsymbol{c} + n_\tau \cdot \upsilon_\tau)$ ▷ Acceleration successor
    instead of the self-loop with rate r, set $\boldsymbol{a} \xrightarrow{\tau} \boldsymbol{a}'$ with rate $n_\tau \cdot r$

It remains to determine the representative for the unbounded interval. In order to avoid infinity, we require an additional input for the analysis, which are deemed upper bounds on possible population of each species. In cases when any upper bound is hard to assume, we can analyse the system with a random one and see if the last interval is reachable with significant probability. If yes, then we need to use this upper bound as a new point in the interval partitioning and try a higher upper bound next time. In general, such conditions can be checked in the abstraction and their violation implies a recommendation to refine the abstract domains accordingly.

Orders-of-Magnitude Abstraction. Such an approximation is thus sufficient to determine most of the time whether the acceleration (sequence of actions) happens sooner or later than e.g. another reaction with rate 10^{-6} or 10^{-2}. Note that this *decision* gets more precise not only as we refine the population levels, but also as the system gets stiffer (the concrete values of the rates differ more), which are normally harder to analyse. For the ease of presentation in our case studies, we shall depict only the magnitude of the rates, i.e. the decadic logarithm rounded to an integer.

Non-determinism and Refinement. If two rates are close to each other, say of the same magnitude (or difference 1), such a rough computation (and rough population discretisation) is not precise enough to determine which of the reactions happens with high probability sooner. Both may be happening roughly at the same pace, or with more information we could conclude one of them is considerably faster. This introduces an uncertainty, showing different behaviours are possible depending on the exact quantities. This indicates points where refinement might be needed if more precise results are required. For instance, with rates of magnitudes 2 and 3, the latter should be happing most of the time, the former only with a few percent chance. If we want to know whether it is rather tens of percent or tenths of percent, we should refine the abstraction.

4 Semi-quantitative Analysis

In this section, we present an approximative analysis technique that describes the most probable transient and steady-state behaviour of the system (also with rough timing) and on demand also the (one or more orders of magnitude) less probable behaviours. As such it is robust in the sense that it is well suited to work with imprecise rates and populations. It is computationally easy (can be done in hand in time required for a computer by other methods), while still yielding significant quantitative results ("in orders of magnitude"). It does not provide exact error guarantees since computing them would be almost as expensive as the classical analysis. It only features trivial limit-style bounds: if the population abstraction gets more and more refined, the probabilities converge to those of the original system; further, the higher the separation between the rate magnitudes, the more precise the approximation is since the other factors (and thus the incurred imprecisions) play less significant role.

Intuitively, the main idea—similar to some multi-rate simulation techniques for stiff systems—is to "simulate" "fast" reactions until the steady state and then examine which slower reactions take place. However, "fast" does not mean faster than some constant, but faster than other transitions in a given state. In other words, we are not distinguishing fast and slow reactions, but tailor this to each state separately. Further, "simulation" is not really a stochastic simulation, but a deterministic choice of the fastest available transition. If a transition is significantly faster than others then this yields what a simulation would yield. When there are transitions with similar rates, e.g. with at most one order of magnitude difference, then both are taken into account as described in the following definition.

Pruned System. Consider the underlying graph of the given CTMC. If we keep only the outgoing transitions with the maximum rate in each state, we call the result *pruned*. If there is always (at most) one transition then the graph consists of several paths leading to cycles. In general when more transitions are kept, it has bottom strongly connected components (bottom SCCs, BSCCs) and some transient parts.

We generalise this concept to *n-pruning* that preserves all transitions with a rate that is not more than n orders of magnitude smaller than the maximum rate in the state. Then the pruning above is 0-pruning, 1-pruning preserves also transitions happening up to 10 times slower, which can thus still happen with dozens of percent, 2-pruning is relevant for analysis where behaviour occurring with units of percent is also tracked etc.

Algorithm Idea. Here we explain the idea of Algorithm 2. The transient parts of the pruned system describe the most probable behaviour from each state until the point where visited states start to repeat a lot (steady state of the pruned system). In the original system, the usual behaviour is then to stay in this SCC C until one of the pruned (slower) reactions occurs, say from state s to state t. This may bring us to a different component of the pruned graph and the analysis process repeats. However, t may also bring us back into C, in which case we stay

in the steady-state, which is basically the same as without the transition from s to t. Further, t might be in the transient part leading to C, in which case these states are added to C and the steady state changes a bit, spreading the distribution slightly also to the previously transient states. Finally, t might be leading us into a component D where this run was previous to visiting C. In that case, the steady-state distribution spreads over all the components visited between D and C, putting a probability mass to each with a different order of magnitude depending on all the (magnitudes of) sojourn times in the transient and steady-state phases on the way.

Using the macros defined in the algorithm, the correctness of the computations can be shown as follows. For the time spent in the transient phase (line 16), we consider the slowest sojourn time on the way times the number of such transitions; this is accurate since the other times are by order(s) of magnitude shorter, hence negligible. The steady-state distribution on a BSCC of the

Algorithm 2. Semi-quantitative analysis

1: $W \leftarrow \emptyset$ ▷ worklist of SCCs to process
2: add {initial state} to W and assign iteration 0 to it ▷ artificial SCC to start the process
3: **while** $W \neq \emptyset$ **do**
4: $C \leftarrow$pop W
▷ Compute and output steady state or its approximation
5: steady-state of C is approximately $minStayingRate/(m \cdot stayingRate(\cdot))$
6: **if** C has no exits **then** continue ▷ definitely bottom SCC, final steady state
▷ Compute and output exiting transitions and the time spent in C
7: $exitStates \leftarrow \arg\min_C(stayingRate(\cdot)/exitingRate(\cdot))$ ▷ Probable exit points
8: $minStayingRate \leftarrow$minimum rate in C, $m \leftarrow$#occurrences there
9: $timeToExit \leftarrow stayingRate(s) \cdot m/(|exitStates| \cdot minStayingRate \cdot exitingRate(s))$
for (arbitrary) $s \in exitStates$
10: **for all** $s \in exitsStates$ **do** ▷ Transient analysis
11: $t \leftarrow$target of the exiting transition
12: $T \leftarrow$SCCs reachable in the pruned graph from t
13: thereby newly reached transitions get assigned iteration of $C + 1$
14: **for** $D \in T$ **do**
▷ Compute and output time to get from t to D
15: $minRate \leftarrow$minimum rate on the way from t to D, $m \leftarrow$#occurrences there
16: $transTime \leftarrow m/minRate$
▷ Determine the new SCC
17: **if** $D = C$ **then** ▷ back to the current SCC
18: add to W the union of C and the new transient path τ from t to C
19: in later steady-state computation, the states of τ will have probability
smaller by a factor of $stayingRate(s)/exitingRate(s)$
20: **else if** D was previously visited **then** ▷ alternating between different SCCs
21: add to W the merge of all SCCs visited between D and C (inclusively)
22: in later steady-state computation, reflect all $timeToExit$ and $transTime$
between D and C
23: **else** ▷ new SCC
24: add D to W

MACROS:
$stayingRate(s)$ is the rate of transitions from s in the pruned graph
$exitingRate(s)$ is the maximum rate of transitions from s not in the pruned graph

pruned graph can be approximated by the $minStayingRate/(m \cdot stayingRate(\cdot))$ on line 5. Indeed, it corresponds to the steady-state distribution if the BSCC is a cycle and the *minStayingRate* significantly larger than other rates in the BSCC since then the return time for the states is approximately $m/minStayingRate$ and the sojourn time $1/stayingRate(\cdot)$. The component is exited from s with the proportion given by its steady-state distribution times the probability to take the exit during that time. The former is approximated above; the latter can be approximated by the density in 0, i.e. by $exitingRate(s)$, since the staying rate is significantly faster. Hence the candidates for exiting are maximising $exitingRate(\cdot)/stayingRate(\cdot)$ as on line 7. There are $|exitStates|$ candidates for exit and the time to exit the component by a particular candidate s is the expected number of visits before exit, i.e. $stayingRate(s) \cdot exitingRate(s)$ times the return time $m \cdot minStayingRate$, hence the expression on line 9.

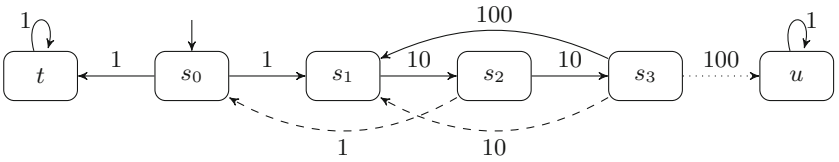


Fig. 2. Alternating transient and steady-state analysis.

For example, consider the system in Fig. 2. Iteration 1 reveals the part with solid lines with two (temporary) BSCCs $\{t\}$ and $\{s_1, s_2, s_3\}$. The former turns out definitely bottom. The latter has a steady state proportional to $(10^{-1}, 10^{-1}, 100^{-1})$. Its most probable exits are the dashed ones, identified in the subsequent iteration 2, probable proportionally to (1/10,10/100); the expected time to take them is $10 \cdot 2/(2 \cdot 10 \cdot 1) = 1 = 100 \cdot 2/(2 \cdot 10 \cdot 10)$. The latter leads back to the current SCC and does not change the set of BSCCs (hence in our examples below we often either skip or merge such iterations for the sake of readability). In contrast, the former leads to a previous SCC; thereafter $\{s_1, s_2, s_3\}$ is no more a bottom SCC and consequently the third exit to u is not even analysed. Nevertheless, it could still happen with minor probability, which can be seen if we consider 1-pruning instead.

5 Experimental Evaluation and Discussion

In order to demonstrate the applicability and accuracy of our approach, we selected the following three biologically relevant case studies. (1) stochastic model of gene expression [22,24], (2) Goutsias's model [23] describing transcription regulation of a repressor protein in bacteriophage λ and (3) viral infection model [43].

Although the underlying CRNs are quite small (up to 5 species and 10 reaction), their analysis is very challenging: (i) the stochasticity has a strong impact

on the dynamics of these systems and thus purely deterministic approximations via ODEs are not accurate, (ii) the systems include species with low, medium, and high populations and thus the resulting state space of the stochastic process is prohibitively large to perform precise numerical analysis and existing reduction/approximation techniques are not sufficient (they are either too imprecise or do not provide sufficient reduction factors), and (iii) the system dynamics leads to bi-modal distributions and/or is affected by stiff reactions.

These models thus represent perfect candidates for evaluating advanced approximation methods including various hybrid approaches [9,24,27]. Although these approaches can handle the models, they typically require tens of minutes or hours of computation time. Similarly simulation-based methods are very time consuming especially in case of very stiff CRN, represented by the viral infection model. We demonstrate that our approach provides accurate predications of the system behaviour and is feasible even when performed manually by a human.

Recall that the algorithm that builds the abstract model of the given CRN takes as input two vectors representing the population discretisation and population bounds. We generally assume that these inputs are provided by users who have a priori knowledge about the system (e.g. in which orders the species population occurs) and that the inputs also reflect the level of details the users are interested in. In the following case studies, we, however, set the inputs only based on the rate orders of the reactions affecting the particular species (unless mentioned otherwise).

5.1 Gene Expression Model

The CRN underlying the gene expression model is described in Table 1. As discussed in [24] and experimentally observed in [18], the system oscillates between two phases characterised by the D_{on} state and the D_{off} state, respectively. Biologists are interested in how the distribution of the D_{on} and D_{off} states is aligned with the distribution of RNA and proteins P, and how the correlation among the distributions depends on the DNA switching rates.

The state vector of the underlying CTMC is given as [P, RNA, D_{off}, D_{on}]. We use very relaxed bounds on the maximal populations, namely the bound 1000 for P and 100 for RNA. Note the DNA invariant $D_{on} + D_{off} = 1$. As in [24], the initial state is given as [10,4,1,0].

We first consider the slow switching rates that lead to a more complicated dynamics including bimodal distributions. In order to demonstrate the refinement step and its effect on the accuracy of the model, we start with a very coarse abstraction. It distinguishes only the zero population and the non-zero populations and thus it is not able to adequately capture the relationship between the DNA state and RNA/P population. The pruned abstract model obtained using Algorithm 1 and 2 is depicted in Fig. 3 (left). The full one before pruning is shown in Fig. 6 [11, Appendix].

The proposed analysis of the model identifies the key trends in the system dynamic. The red transitions, representing iterations 1–3, capture the most probable paths in the system. The green component includes states with DNA on

Table 1. Gene expression. For slow DNA switching, $r_1 = r_2 = 0.05$. For fast DNA switching, $r_1 = r_2 = 1$. The rates are in h^{-1}.

$D_{off} \xrightarrow{r_1} D_{on}$ $\quad D_{on} \xrightarrow{r_2} D_{off}$ $\quad D_{on} \xrightarrow{10} D_{on} + RNA$ $\quad RNA \xrightarrow{1} \emptyset$

$RNA \xrightarrow{4} RNA + P$ $\quad P \xrightarrow{1} \emptyset$ $\quad P + D_{off} \xrightarrow{0.0015} P + D_{on}$

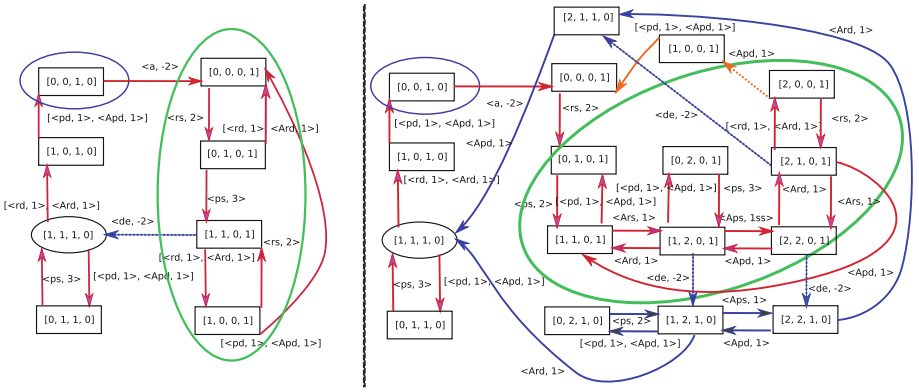


Fig. 3. Pruned abstraction for the gene expression model using the coarse population discretisation (left) and after the refinement (right). The state vector is [P, RNA, D_{off}, D_{on}].

(i.e. $D_{on} = 1$) where the system oscillates. The component is reached via the blue state with D_{off} and no RNAs/P. The blue state is promptly reached from the initial state and then the system waits (roughly 100 h according our rate abstraction) for the next DNA activation. The oscillation is left via a deactivation in the iteration 4 (the blue dotted transition)[5]. The estimation of the exit time computed using Algorithm 2 is also 100 h. The deactivation is then followed by fast red transitions leading to the blue state, where the system waits for the next activation. Therefore, we obtain an oscillation between the blue state and the green component, representing the expected oscillation between the D_{on} and D_{off} states.

As expected, this abstraction does not clearly predict the bimodal distribution on the RNA/P populations as the trivial population levels do not bear any information beside reaction enabledness. In order to obtain a more accurate analysis of the system, we refine the population discretisation using a single level threshold for P and DNA, that is equal to 100 and 10, respectively (the rates in the CRN indicate that the population of P reaches higher values).

Figure 3 (right) depicts the pruned abstract model with the new discretisation (the full model is depicted in Fig. 7 [11, Appendix]. We again obtain the oscillation between the green component representing DNA_{on} states and the blue DNA_{off} state. The states in the green component more accurately predicts

[5] In Fig. 3, the dotted transitions denote exit transitions representing the deactivations.

that in the DNA_{on} states the populations of RNA and P are high and drop to zero only for short time periods. The figure also shows orange transitions within the iteration 2 that extend the green component by two states. Note that the system promptly returns from these states back to the green component. After the deactivation in the iteration 4, the system takes (within the same iteration) the fast transitions (solid blue) leading to the blue component where system waits for another activation and where the mRNA/protein populations decrease. The expected time spent in states on blue solid transitions is small and thus we can reliably predict the bimodal distribution of the mRNA/P populations and its correlation with the DNA state. The refined abstraction also reveals that the switching time from the DNA_{on} mode to the DNA_{off} mode is lower. These predications are in accordance with the results obtained in [24]. See Fig. 8 [11, Appendix] that is adopted from [24] and illustrates these results.

To further test the accuracy of our approach, we consider the fast switching between the DNA states. We follow the study in [24] and increase the rates by two orders of magnitude. We use the refined population discretisation and obtain a very similar abstraction as in Fig. 3 (right). We again obtain the oscillation between the green component (DNA_{on} states and nonzero RNA/protein populations) and the blue state (DNA_{off} and zero RNA/protein populations). The only difference is in fact the transition rates corresponding to the activation and deactivation causing that the switching rate between the components is much faster. As a consequence, the system spends a longer period in the blue transient states with D_{off} and nonzero RNA/protein populations. The time spent in these states decreases the correlation between the DNA state and the RNA/protein populations as well as the bimodality in the population distribution. This is again in the accordance with [24].

To conclude this case study, we observe a very aligned agreement between the results obtained using our approach and results in [24] obtained via advanced and time consuming numerical methods. We would like to emphasise that our abstraction and its solution is obtained within a fraction of a second while the numerical methods have to approximate solutions of equations describing high-order conditional moments of the population distributions. As [24] does not report the runtime of the analysis and the implementation of their methods is not publicly available, we cannot directly compare the time complexity.

5.2 Goutsias's Model

Goutsias's model illustrated in Table 2 is widely used for evaluation of various numerical and simulation based techniques. As showed e.g. in [23], the system has with a high probability the following transient behaviour. In the first phase, the system switches with a high rate between the non-active DNA (denoted DNA) and the active DNA (DNA.D). During this phase the population of RNA, monomers (M) and dimers (D) gradually increase (with only negligible oscillations). After around 15 min, the DNA is blocked (DNA.2D) and the population of RNA decreases while the population of M and D is relatively stable. After all RNA degrades (around another 15 min) the system switches to the third

Table 2. Goutsias' Model. The rates are in s^{-1}

$RNA \xrightarrow{0.043} RNA + M$	$M \xrightarrow{7\times10^{-4}} \emptyset$	$RNA \xrightarrow{4\times10^{-3}} \emptyset$
$DNA + D \xrightarrow{0.002} DNA.D$	$DNA.D \xrightarrow{0.48} DNA + D$	
$DNA.D + D \xrightarrow{2\times10^{-4}} DNA.2D$	$M+M \xrightarrow{0.083} D$	$D \xrightarrow{0.5} M + M$
$DNA.2D \xrightarrow{9\times10^{-12}} DNA.D + D$	$DNA.D \xrightarrow{0.072} RNA + DNA.D$	

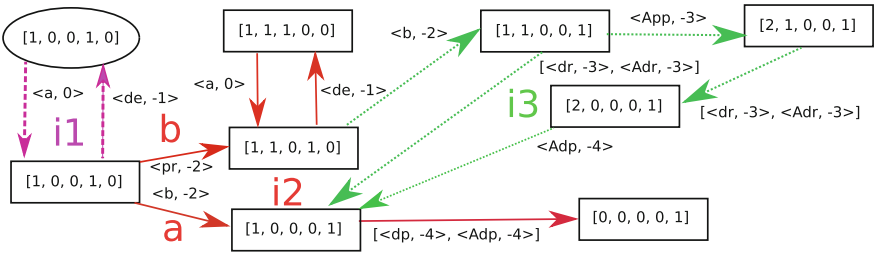


Fig. 4. Pruned abstraction for the Goutsias' model. The state vector is [M + D, RNA, DNA, DNA.D, DNA.2D]

phase where the population of M and D slowly decreases. Further, there is a non-negligible probability that the DNA is blocked at the beginning while the population of RNA is still small and the system promptly dies out.

Although the system is quite suitable for the hybrid approaches (there is no strong bimodality and only a limited stiffness), the analysis still takes 10 to 50 min depending on the required precision [27]. We demonstrate that our approach is able to accurately predict the main transient behaviour as well as the non-negligible probability that the system promptly dies out.

The state vector is given as [M, D, RNA, DNA, DNA.D, DNA.2D] and the initial state is set to [2, 6, 0, 1, 0, 0] as in [27]. We start our analysis with a coarse population discretisation with a single threshold 100 for M and D and a single threshold 10 for RNA. We relax the bounds, in particular, 1000 for M and D, and 100 for RNA. Note that these numbers were selected solely based on the rate orders of the relevant reactions. Note the DNA invariant DNA + DNA.D + DNA.2D = 1.

Figure 4 illustrates the pruned abstract model we obtained (the full model is depicted in Fig. 9 [11, Appendix]. For a better visualisation, we merged the state components corresponding to M and D into one component with M + D. As there is the fast reversible dimerisation, the actual distributions between the population of M and D does not affect the transient behaviour we are interested in.

The analysis of the model shows the following transient behaviour. The purple dotted loop in the iteration `i1` represents (de-)activation of the DNA. The expected exit time of this loop is 100 s. According to our abstraction, there are two options (with the same probability) to exit the loop: (1) the path `a` rep-

resents the DNA blocking followed by the quick extinction and (2) the path b corresponds to the production of RNA and its followed by the red loop in the i2 that again represents (de-)activation of the DNA. Note that according our abstraction, this loop contains states with the populations of M/D as well as RNA up to 100 and 10, respectively.

The expected exit time of this loop is again 100 s and there are two options how to leave the loop: (1) the path within the iteration $i3$ (taken with roughly 90%) represents again the DNA blocking and it is followed by the extension of RNA and consequently by the extension of M/D in about 1000 s and (2) the path within the iteration 5 (shown in the full graph in Fig. 9 [11, Appendix]) taken with roughly 10% represents the series of protein productions and leads to the states with a high number of proteins (above 100 in our population discretisation). Afterwards, there is again a series of DNA (de-)activations followed by the DNA blocking and the extinction of RNA. As before, this leads to the extinction of M/D in about 1000 s.

Although this abstraction already shows the transient behaviour leading to the extinction in about 30 min, it introduces the following inaccuracy with respect to the known behaviour: (1) the probability of the fast extinction is higher and (2) we do not observe the clear bell-shape pattern on the RNA (i.e. the level 2 for the RNA is not reached in the abstraction). As in the previous case study, the problem is that the population discretisation is too coarse. It causes that the total rate of the DNA blocking (affected by the M/D population via the mass action kinetics) is too high in the states with the M/D population level 1. This can be directly seen in the interval CTMC representation where the rate spans many orders of magnitude, incurring too much imprecision. The refinement of the M/D population discretisation eliminates the first inaccuracy. To obtain the clear bell-shape patter on RNA, one has to refine also the RNA population discretisation.

5.3 Viral Infection

The viral infection model described in Table 3 represents the most challenging system we consider. It is highly stochastic, extremely stiff, with all species presenting high variance and some also very high molecular populations. Moreover, there is a bimodal distribution on the RNA population. As a consequence, the solution of the full CME, even using advanced reduction and aggregation techniques, is prohibitive due to state-space explosion and stochastic simulation are very time consuming. State-of-the-art hybrid approaches integrating the LNA and an adaptive population partitioning [9] can handle this system but also need a very long execution time. For example, a transient analysis up to time $t = 50$ requires around 20 min and up to $t = 200$ more than an hour.

To evaluate the accuracy of our approach on this challenging model, we also focus on the same transient analysis, namely, we are interested in the distribution of RNA at time $t = 200$. The analysis in [9] predicts a bimodal distribution where, the probability that RNA is zero in around 20% and the remaining probability has Gaussian distribution with mean around 17 and the probability that there

Table 3. Viral Infection. The rates are day^{-1}

$$\mathrm{DNA} + \mathrm{P} \xrightarrow{7\times10^{-6}} \mathrm{V} \qquad \mathrm{DNA} \xrightarrow{0.025} \mathrm{DNA} + \mathrm{R} \qquad \mathrm{RNA} \xrightarrow{0.25} \emptyset$$
$$\mathrm{RNA} \xrightarrow{1} \mathrm{RNA} + \mathrm{DNA} \qquad \mathrm{RNA} \xrightarrow{1000} \mathrm{RNA.} + \mathrm{P} \qquad \mathrm{P} \xrightarrow{1.99} \emptyset$$

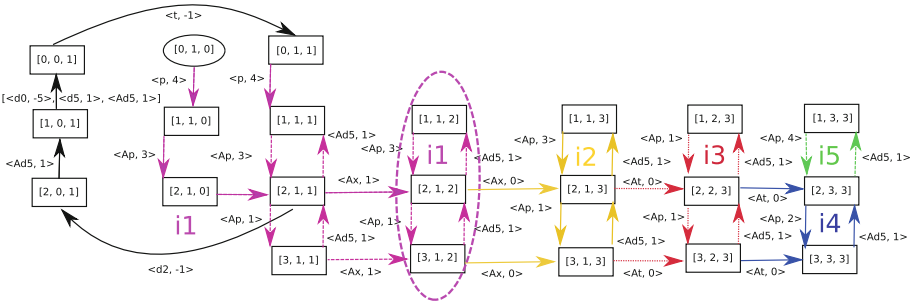


Fig. 5. Pruned abstraction for the viral infection model. The state vector is [P, RNA, DNA].

is more than 30 RNAs is close to zero. This is confirmed by simulation-based analysis in [23] showing also the gradual growth of the RNA population. The simulation-based analysis in [43], however, estimates a lower probability (around 3%) that RNA is 0 and higher mean of the remaining Gaussian distribution (around 23). Recall that obtaining accurate results using simulations is extremely time consuming due to very stiff reactions (a single simulation for $t = 200$ takes around 20 s).

In the final experiments, we analyse the distribution of RNA at time $t = 200$ using our approach. The state vector is given as [P, RNA, DNA] and we start with the concrete state [0, 1, 0]. To sufficiently reason about the RNA population and to handle the very high population of the proteins, we use the following population discretisation: thresholds {10, 1000} for P, {10, 30} for RNA, and {10, 100} for DNA. As before, we use very relaxed bounds 10000, 100, and 1000 for P, RNA, and D, respectively. Note that we ignore the population of the virus V as it does not affect the dynamics of the other species. This simplification makes the visualisation of our approach more readable and has no effect on the complexity of the analysis.

Figure 5 illustrates the obtained abstract model enabling the following transient analysis (the full model is depicted in Fig. 10 [11, Appendix]. In a few days the system reaches from the initial state the loop (depicted by the purple dashed ellipse) within the iteration *i1*. The loop includes states where RNA has level 1, DNA has level 2 and P oscillates between the levels 2 and 3. Before entering the loop, there is a non-negligible probability (orders of percent) that the RNA drops to 0 via the full black branch that returns to transient part of the loop in *i1*. In this branch the system can also die out (not shown in this figure, see the full model) with probability in the order of tenths of percent.

The average exit time of the loop in *i1* is in the order of 10 days and the system goes to the yellow loop within the iteration *i2*, where the DNA level is increased to 3 (RNA level is unchanged and P again oscillates between the levels 2 and 3). The average exit time of the loop in *i2* is again in the order of 10 days and systems goes to the dotted red loop within iteration *i3*. The transition represents the sequence of RNA synthesis that leads to RNA level 2. P oscillates as before. Finally, the system leaves the loop in *i3* (this takes another dozen days) and reaches RNA level 3 in iterations *i4* and *i5* where the DNA level remains at the level 3 and P oscillates. The iteration *i4* and *i5* thus roughly correspond to the examined transient time $t = 200$.

The analysis clearly demonstrates that our approach leads to the behaviour that is well aligned with the previous experiments. We observed growth of the RNA population with a non-negligible probability of its extinction. The concrete quantities (i.e. the probability of the extinction and the mean RNA population) are closer to the analysis in [43]. The quantities are indeed affected by the population discretisation and can be further refined. We would like to emphasise that in contrast to the methods presented in [9,23,43] requiring hours of intensive numerical computation, our approach can be done even manually on the paper.

References

1. Abate, A., Katoen, J.P., Lygeros, J., Prandini, M.: Approximate model checking of stochastic hybrid systems. Eur. J. Control **16**, 624–641 (2010)
2. Abate, A., Brim, L., Češka, M., Kwiatkowska, M.: Adaptive aggregation of Markov chains: quantitative analysis of chemical reaction networks. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 195–213. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_12
3. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. **20**(4), 279–304 (2007)
4. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
5. Bortolussi, L., Hillston, J.: Fluid model checking. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 333–347. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32940-1_24
6. Buchholz, P.: Exact performance equivalence: an equivalence relation for stochastic automata. Theor. Comput. Sci. **215**(1–2), 263–287 (1999)
7. Cao, Y., Gillespie, D.T., Petzold, L.R.: The slow-scale stochastic simulation algorithm. J. Chem. Phys. **122**(1), 014116 (2005)
8. Cardelli, L.: Two-domain DNA strand displacement. Math. Struct. Comput. Sci. **23**(02), 247–271 (2013)
9. Cardelli, L., Kwiatkowska, M., Laurenti, L.: A stochastic hybrid approximation for chemical kinetics based on the linear noise approximation. In: Bartocci, E., Lio, P., Paoletti, N. (eds.) CMSB 2016. LNCS, vol. 9859, pp. 147–167. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45177-0_10
10. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Maximal aggregation of polynomial dynamical systems. Proc. Natl. Acad. Sci. **114**(38), 10029–10034 (2017)

11. Češka, M., Křetínský, J.: Semi-quantitative abstraction and analysis of chemical reaction networks. Technical report abs/1905.09914, arXiv.org (2019)
12. Chellaboina, V., Bhat, S.P., Haddad, W.M., Bernstein, D.S.: Modeling and analysis of mass-action kinetics. IEEE Control Syst. Mag. **29**(4), 60–78 (2009)
13. Desharnais, J., Laviolette, F., Tracol, M.: Approximate analysis of probabilistic processes: logic, simulation and games. In: Quantitative Evaluation of SysTems (QEST), pp. 264–273. IEEE (2008)
14. D'Innocenzo, A., Abate, A., Katoen, J.P.: Robust PCTL model checking. In: Hybrid Systems: Computation and Control (HSCC), pp. 275–285. ACM (2012)
15. Engblom, S.: Computing the moments of high dimensional solutions of the master equation. Appl. Math. Comput. **180**(2), 498–515 (2006)
16. Ethier, S.N., Kurtz, T.G.: Markov Processes: Characterization and Convergence, vol. 282. Wiley, New York (2009)
17. Ferm, L., Lötstedt, P.: Adaptive solution of the master equation in low dimensions. Appl. Numer. Math. **59**(1), 187–204 (2009)
18. Gandhi, S.J., Zenklusen, D., Lionnet, T., Singer, R.H.: Transcription of functionally related constitutive genes is not coordinated. Nat. Struct. Mol. Biol. **18**(1), 27 (2011)
19. Ganguly, A., Altintan, D., Koeppl, H.: Jump-diffusion approximation of stochastic reaction dynamics: error bounds and algorithms. Multiscale Model. Simul. **13**(4), 1390–1419 (2015)
20. Giacobbe, M., Guet, C.C., Gupta, A., Henzinger, T.A., Paixão, T., Petrov, T.: Model checking gene regulatory networks. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 469–483. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_47
21. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2361 (1977)
22. Golding, I., Paulsson, J., Zawilski, S.M., Cox, E.C.: Real-time kinetics of gene activity in individual bacteria. Cell **123**(6), 1025–1036 (2005)
23. Goutsias, J.: Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems. J. Chem. Phys. **122**(18), 184102 (2005)
24. Hasenauer, J., Wolf, V., Kazeroonian, A., Theis, F.: Method of conditional moments (MCM) for the chemical master equation. J. Math. Biol. 1–49 (2013). https://doi.org/10.1007/s00285-013-0711-5
25. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theor. Comput. Sci. **391**(3), 239–257 (2008)
26. Henzinger, T.A., Mateescu, M., Wolf, V.: Sliding window abstraction for infinite Markov chains. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 337–352. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_27
27. Henzinger, T.A., Mikeev, L., Mateescu, M., Wolf, V.: Hybrid numerical solution of the chemical master equation. In: Computational Methods in Systems Biology (CMSB), pp. 55–65. ACM (2010)
28. Hepp, B., Gupta, A., Khammash, M.: Adaptive hybrid simulations for multiscale stochastic reaction networks. J. Chem. Phys. **142**(3), 034118 (2015)
29. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)
30. Katoen, J.-P., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for continuous-time Markov chains. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 311–324. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_37

31. Kwiatkowska, M., Thachuk, C.: Probabilistic model checking for biology. Softw. Syst. Saf. **36**, 165 (2014)
32. Lakin, M.R., Parker, D., Cardelli, L., Kwiatkowska, M., Phillips, A.: Design and analysis of DNA strand displacement devices using probabilistic model checking. J. R. Soc. Interface **9**(72), 1470–1485 (2012)
33. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Inf. Comput. **94**(1), 1–28 (1991)
34. Madsen, C., Myers, C., Roehner, N., Winstead, C., Zhang, Z.: Utilizing stochastic model checking to analyze genetic circuits. In: Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), pp. 379–386. IEEE (2012)
35. Mateescu, M., Wolf, V., Didier, F., Henzinger, T.A.: Fast adaptive uniformization of the chemical master equation. IET Syst. Biol. **4**(6), 441–452 (2010)
36. Munsky, B., Khammash, M.: The finite state projection algorithm for the solution of the chemical master equation. J. Chem. Phys. **124**, 044104 (2006)
37. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
38. Rao, C.V., Arkin, A.P.: Stochastic chemical kinetics and the quasi-steady-state assumption: application to the Gillespie algorithm. J. Chem. Phys. **118**(11), 4999–5010 (2003)
39. Salis, H., Kaznessis, Y.: Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. J. Chem. Phys. **122**(5), 054103 (2005)
40. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Natl. Acad. Sci. U. S. A. **107**(12), 5393–5398 (2010)
41. Soudjani, S.E.Z., Abate, A.: Adaptive and sequential gridding procedures for the abstraction and verification of stochastic processes. SIAM J. Appl. Dyn. Syst. **12**(2), 921–956 (2013)
42. Esmaeil Zadeh Soudjani, S., Abate, A.: Precise approximations of the probability distribution of a Markov process in time: an application to probabilistic invariance. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 547–561. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_45
43. Srivastava, R., You, L., Summers, J., Yin, J.: Stochastic vs. deterministic modeling of intracellular viral kinetics. J. Theor. Biol. **218**(3), 309–321 (2002)
44. Van Kampen, N.G.: Stochastic Processes in Physics and Chemistry, vol. 1. Elsevier, New York (1992)
45. Zhang, J., Watson, L.T., Cao, Y.: Adaptive aggregation method for the chemical master equation. Int. J. Comput. Biol. Drug Des. **2**(2), 134–148 (2009)

PAC Statistical Model Checking for Markov Decision Processes and Stochastic Games

Pranav Ashok, Jan Křetínský, and Maximilian Weininger(✉)

Technical University of Munich, Munich, Germany
maxi.weininger@tum.de

Abstract. Statistical model checking (SMC) is a technique for analysis of probabilistic systems that may be (partially) unknown. We present an SMC algorithm for (unbounded) reachability yielding probably approximately correct (PAC) guarantees on the results. We consider both the setting (i) with no knowledge of the transition function (with the only quantity required a bound on the minimum transition probability) and (ii) with knowledge of the topology of the underlying graph. On the one hand, it is the first algorithm for stochastic games. On the other hand, it is the first practical algorithm even for Markov decision processes. Compared to previous approaches where PAC guarantees require running times longer than the age of universe even for systems with a handful of states, our algorithm often yields reasonably precise results within minutes, not requiring the knowledge of mixing time.

1 Introduction

Statistical model checking (SMC) [YS02a] is an analysis technique for probabilistic systems based on

1. simulating finitely many finitely long runs of the system,
2. statistical analysis of the obtained results,
3. yielding a confidence interval/probably approximately correct (PAC) result on the probability of satisfying a given property, i.e., there is a non-zero probability that the bounds are incorrect, but they are correct with probability that can be set arbitrarily close to 1.

One of the advantages is that it can avoid the state-space explosion problem, albeit at the cost of weaker guarantees. Even more importantly, this technique is applicable even when the model is not known (*black-box* setting) or only

This research was funded in part by TUM IGSSE Grant 10.06 (PARSEC), the Czech Science Foundation grant No. 18-11193S, and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification".

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 497–519, 2019.
https://doi.org/10.1007/978-3-030-25540-4_29

qualitatively known (*grey-box* setting), where the exact transition probabilities are unknown such as in many cyber-physical systems.

In the basic setting of Markov chains [Nor98] with (time- or step-)bounded properties, the technique is very efficient and has been applied to numerous domains, e.g. biological [JCL+09,PGL+13], hybrid [ZPC10,DDL+12,EGF12, Lar12] or cyber-physical [BBB+10,CZ11,DDL+13] systems and a substantial tool support is available [JLS12,BDL+12,BCLS13,BHH12]. In contrast, whenever either (i) infinite time-horizon properties, e.g. reachability, are considered or (ii) non-determinism is present in the system, providing any guarantees becomes significantly harder.

Firstly, for *infinite time-horizon properties* we need a stopping criterion such that the infinite-horizon property can be reliably evaluated based on a finite prefix of the run yielded by simulation. This can rely on the the complete knowledge of the system (*white-box* setting) [YCZ10,LP08], the topology of the system (grey box) [YCZ10,HJB+10], or a lower bound $p_{\min}$ on the minimum transition probability in the system (black box) [DHKP16,BCC+14].

Secondly, for Markov decision processes (MDP) [Put14] with (non-trivial) *non-determinism*, [HMZ+12] and [LP12] employ reinforcement learning [SB98] in the setting of bounded properties or discounted (and for the purposes of approximation thus also bounded) properties, respectively. The latter also yields PAC guarantees.

Finally, for MDP with unbounded properties, [BFHH11] deals with MDP with spurious non-determinism, where the way it is resolved does not affect the desired property. The general non-deterministic case is treated in [FT14, BCC+14], yielding PAC guarantees. However, the former requires the knowledge of mixing time, which is at least as hard to compute; the algorithm in the latter is purely theoretical since before a single value is updated in the learning process, one has to simulate longer than the age of universe even for a system as simple as a Markov chain with 12 states having at least 4 successors for some state.

Our contribution is an SMC algorithm with PAC guarantees for (i) MDP and unbounded properties, which runs for realistic benchmarks [HKP+19] and confidence intervals in orders of minutes, and (ii) is the first algorithm for stochastic games (SG). It relies on different techniques from literature.

1. The increased practical performance rests on two pillars:
   - extending early detection of bottom strongly connected components in Markov chains by [DHKP16] to end components for MDP and simple end components for SG;
   - improving the underlying PAC Q-learning technique of [SLW+06]:
     (a) learning is now model-based with better information reuse instead of model-free, but in realistic settings with the same memory requirements,
     (b) better guidance of learning due to interleaving with precise computation, which yields more precise value estimates.
     (c) splitting confidence over all relevant transitions, allowing for variable width of confidence intervals on the learnt transition probabilities.

2. The transition from algorithms for MDP to SG is possible via extending the over-approximating value iteration from MDP [BCC+14] to SG by [KKKW18].

To summarize, we give an anytime PAC SMC algorithm for (unbounded) reachability. It is the first such algorithm for SG and the first practical one for MDP.

Related Work

Most of the previous efforts in SMC have focused on the analysis of properties with *bounded* horizon [YS02a, SVA04, YKNP06, JCL+09, JLS12, BDL+12].

SMC of *unbounded* properties was first considered in [HLMP04] and the first approach was proposed in [SVA05], but observed incorrect in [HJB+10]. Notably, in [YCZ10] two approaches are described. The first approach proposes to terminate sampled paths at every step with some probability p_{term} and reweight the result accordingly. In order to guarantee the asymptotic convergence of this method, the second eigenvalue λ of the chain and its mixing time must be computed, which is as hard as the verification problem itself and requires the complete knowledge of the system (white box setting). The correctness of [LP08] relies on the knowledge of the second eigenvalue λ, too. The second approach of [YCZ10] requires the knowledge of the chain's topology (grey box), which is used to transform the chain so that all potentially infinite paths are eliminated. In [HJB+10], a similar transformation is performed, again requiring knowledge of the topology. In [DHKP16], only (a lower bound on) the minimum transition probability $p_{\min}$ is assumed and PAC guarantees are derived. While unbounded properties cannot be analyzed without any information on the system, knowledge of $p_{\min}$ is a relatively light assumption in many realistic scenarios [DHKP16]. For instance, bounds on the rates for reaction kinetics in chemical reaction systems are typically known; for models in the PRISM language [KNP11], the bounds can be easily inferred without constructing the respective state space. In this paper, we thus adopt this assumption.

In the case with general *non-determinism*, one approach is to give the nondeterminism a probabilistic semantics, e.g., using a uniform distribution instead, as for timed automata in [DLL+11a, DLL+11b, Lar13]. Others [LP12, HMZ+12, BCC+14] aim to quantify over all strategies and produce an ϵ-optimal strategy. In [HMZ+12], candidates for optimal strategies are generated and gradually improved, but "at any given point we cannot quantify how close to optimal the candidate scheduler is" (cited from [HMZ+12]) and the algorithm "does not in general converge to the true optimum" (cited from [LST14]). Further, [LST14, DLST15, DHS18] randomly sample compact representation of strategies, resulting in useful lower bounds if ε-schedulers are frequent. [HPS+19] gives a convergent model-free algorithm (with no bounds on the current error) and identifies that the previous [SKC+14] "has two faults, the second of which also affects approaches [...] [HAK18, HAK19]".

Several approaches provide SMC for MDPs and unbounded properties with *PAC guarantees*. Firstly, similarly to [LP08, YCZ10], [FT14] requires (1) the

mixing time T of the MDP. The algorithm then yields PAC bounds in time polynomial in T (which in turn can of course be exponential in the size of the MDP). Moreover, the algorithm requires (2) the ability to restart simulations also in non-initial states, (3) it only returns the strategy once all states have been visited (sufficiently many times), and thus (4) requires the size of the state space $|S|$. Secondly, [BCC+14], based on delayed Q-learning (DQL) [SLW+06], lifts the assumptions (2) and (3) and instead of (1) mixing time requires only (a bound on) the minimum transition probability $p_{\min}$. Our approach additionally lifts the assumption (4) and allows for running times faster than those given by T, even without the knowledge of T.

Reinforcement learning (without PAC bounds) for stochastic games has been considered already in [LN81,Lit94,BT99]. [WT16] combines the special case of almost-sure satisfaction of a specification with optimizing quantitative objectives. We use techniques of [KKKW18], which however assumes access to the transition probabilities.

2 Preliminaries

2.1 Stochastic Games

A *probability distribution* on a finite set X is a mapping $\delta : X \to [0,1]$, such that $\sum_{x \in X} \delta(x) = 1$. The set of all probability distributions on X is denoted by $\mathcal{D}(X)$. Now we define turn-based two-player stochastic games. As opposed to the notation of e.g. [Con92], we do not have special stochastic nodes, but rather a probabilistic transition function.

Definition 1 (SG). *A* stochastic game (SG) is a tuple $\mathsf{G} = (\mathsf{S}, \mathsf{S}_\Box, \mathsf{S}_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \mathbb{T})$, where S is a finite set of *states* partitioned[1] into the sets $\mathsf{S}_\Box$ and $\mathsf{S}_\bigcirc$ of states of the player *Maximizer* and *Minimizer*[2], respectively $\mathsf{s}_0 \in \mathsf{S}$ is the *initial* state, A is a finite set of *actions*, $\mathsf{Av} : \mathsf{S} \to 2^{\mathsf{A}}$ assigns to every state a set of *available* actions, and $\mathbb{T} : \mathsf{S} \times \mathsf{A} \to \mathcal{D}(\mathsf{S})$ is a *transition function* that given a state s and an action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ yields a probability distribution over *successor* states. Note that for ease of notation we write $\mathbb{T}(\mathsf{s}, \mathsf{a}, \mathsf{t})$ instead of $\mathbb{T}(\mathsf{s}, \mathsf{a})(\mathsf{t})$.

A Markov decision process (MDP) is a special case of SG where $\mathsf{S}_\bigcirc = \emptyset$. A Markov chain (MC) can be seen as a special case of an MDP, where for all $\mathsf{s} \in \mathsf{S} : |\mathsf{Av}(\mathsf{s})| = 1$. We assume that SG are non-blocking, so for all states s we have $\mathsf{Av}(\mathsf{s}) \neq \emptyset$.

For a state s and an available action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$, we denote the set of successors by $\mathsf{Post}(\mathsf{s}, \mathsf{a}) := \{\mathsf{t} \mid \mathbb{T}(\mathsf{s}, \mathsf{a}, \mathsf{t}) > 0\}$. We say a state-action pair (s, a) is an *exit* of a set of states T, written (s, a) exits T, if $\exists \mathsf{t} \in \mathsf{Post}(\mathsf{s}, \mathsf{a}) : \mathsf{t} \notin T$, i.e., if with some probability a successor outside of T could be chosen.

We consider algorithms that have a limited information about the SG.

[1] I.e., $\mathsf{S}_\Box \subseteq \mathsf{S}$, $\mathsf{S}_\bigcirc \subseteq \mathsf{S}$, $\mathsf{S}_\Box \cup \mathsf{S}_\bigcirc = \mathsf{S}$, and $\mathsf{S}_\Box \cap \mathsf{S}_\bigcirc = \emptyset$.

[2] The names are chosen, because Maximizer maximizes the probability of reaching a given target state, and Minimizer minimizes it.

Definition 2 (Black box and grey box). *An algorithm inputs an SG as* black box *if it cannot access the whole tuple, but*

- *it knows the initial state,*
- *for a given state, an oracle returns its player and available action,*
- *given a state* s *and action* a*, it can sample a successor* t *according to* $\mathbb{T}(\mathsf{s},\mathsf{a})$*,*[3]
- *it knows* $p_{\min} \leq \min_{\substack{\mathsf{s}\in\mathsf{S},\mathsf{a}\in\mathsf{Av}(\mathsf{s}) \\ \mathsf{t}\in\mathsf{Post}(\mathsf{s},\mathsf{a})}} \mathbb{T}(\mathsf{s},\mathsf{a},\mathsf{t})$*, an under-approximation of the minimum transition probability.*

When input as grey box *it additionally knows the number* $|\mathsf{Post}(\mathsf{s},\mathsf{a})|$ *of successors for each state* s *and action* a*.*[4]

The semantics of SG is given in the usual way by means of strategies and the induced Markov chain [BK08] and its respective probability space, as follows. An *infinite path* ρ is an infinite sequence $\rho = \mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1 \cdots \in (\mathsf{S}\times\mathsf{A})^\omega$, such that for every $i \in \mathbb{N}$, $\mathsf{a}_i \in \mathsf{Av}(\mathsf{s}_i)$ and $\mathsf{s}_{i+1} \in \mathsf{Post}(\mathsf{s}_i,\mathsf{a}_i)$.

A *strategy* of Maximizer or Minimizer is a function $\sigma : \mathsf{S}_\square \to \mathcal{D}(\mathsf{A})$ or $\mathsf{S}_\bigcirc \to \mathcal{D}(\mathsf{A})$, respectively, such that $\sigma(\mathsf{s}) \in \mathcal{D}(\mathsf{Av}(\mathsf{s}))$ for all s. Note that we restrict to memoryless/positional strategies, as they suffice for reachability in SGs [CH12].

A pair (σ,τ) of strategies of Maximizer and Minimizer induces a Markov chain $\mathsf{G}^{\sigma,\tau}$ with states S, s_0 being initial, and the transition function $\mathbb{T}(\mathsf{s})(\mathsf{t}) = \sum_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})} \sigma(\mathsf{s})(\mathsf{a}) \cdot \mathbb{T}(\mathsf{s},\mathsf{a},\mathsf{t})$ for states of Maximizer and analogously for states of Minimizer, with σ replaced by τ. The Markov chain induces a unique probability distribution $\mathbb{P}^{\sigma,\tau}$ over measurable sets of infinite paths [BK08, Ch. 10].

2.2 Reachability Objective

For a goal set $\mathsf{Goal} \subseteq \mathsf{S}$, we write $\Diamond\mathsf{Goal} := \{\mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1\cdots \mid \exists i \in \mathbb{N} : \mathsf{s}_i \in \mathsf{Goal}\}$ to denote the (measurable) set of all infinite paths which eventually reach Goal. For each $\mathsf{s} \in \mathsf{S}$, we define the *value* in s as

$$\mathsf{V}(\mathsf{s}) := \sup_\sigma \inf_\tau \mathbb{P}_s^{\sigma,\tau}(\Diamond\mathsf{Goal}) = \inf_\tau \sup_\sigma \mathbb{P}_s^{\sigma,\tau}(\Diamond\mathsf{Goal}),$$

where the equality follows from [Mar75]. We are interested in $\mathsf{V}(\mathsf{s}_0)$, its ε-approximation and the corresponding (ε-)optimal strategies for both players.

[3] Up to this point, this definition conforms to black box systems in the sense of [SVA04] with sampling from the initial state, being slightly stricter than [YS02a] or [RP09], where simulations can be run from any desired state. Further, we assume that we can choose actions for the adversarial player or that she plays fairly. Otherwise the adversary could avoid playing her best strategy during the SMC, not giving SMC enough information about her possible behaviours.

[4] This requirement is slightly weaker than the knowledge of the whole topology, i.e. $\mathsf{Post}(\mathsf{s},\mathsf{a})$ for each s and a.

Let Zero be the set of states, from which there is no finite path to any state in Goal. The value function V satisfies the following system of equations, which is referred to as the *Bellman equations*:

$$\mathsf{V}(\mathsf{s}) = \begin{cases} \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s},\mathsf{a}) & \text{if } \mathsf{s} \in \mathsf{S}_{\square} \\ \min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s},\mathsf{a}) & \text{if } \mathsf{s} \in \mathsf{S}_{\bigcirc} \\ 1 & \text{if } \mathsf{s} \in \mathsf{Goal} \\ 0 & \text{if } \mathsf{s} \in \mathsf{Zero} \end{cases}$$

with the abbreviation $\mathsf{V}(\mathsf{s},\mathsf{a}) := \sum_{s' \in S} \mathbb{T}(\mathsf{s},\mathsf{a},\mathsf{s}') \cdot \mathsf{V}(\mathsf{s}')$. Moreover, V is the *least* solution to the Bellman equations, see e.g. [CH08].

2.3 Bounded and Asynchronous Value Iteration

The well known technique of value iteration, e.g. [Put14,RF91], works by starting from an under-approximation of value function and then applying the Bellman equations. This converges towards the least fixpoint of the Bellman equations, i.e. the *value function*. Since it is difficult to give a convergence criterion, the approach of bounded value iteration (BVI, also called interval iteration) was developed for MDP [BCC+14,HM17] and SG [KKKW18]. Beside the under-approximation, it also updates an over-approximation according to the Bellman equations. The most conservative over-approximation is to use an upper bound of 1 for every state. For the under-approximation, we can set the lower bound of target states to 1; all other states have a lower bound of 0. We use the function INITIALIZE_BOUNDS in our algorithms to denote that the lower and upper bounds are set as just described; see [AKW19, Algorithm 8] for the pseudocode. Additionally, BVI ensures that the over-approximation converges to the least fixpoint by taking special care of *end components*, which are the reason for not converging to the true value from above.

Definition 3 (End component (EC)). *A non-empty set $T \subseteq \mathsf{S}$ of states is an* end component (EC) *if there is a non-empty set $B \subseteq \bigcup_{\mathsf{s} \in T} \mathsf{Av}(s)$ of actions such that (i) for each $\mathsf{s} \in T, \mathsf{a} \in B \cap \mathsf{Av}(\mathsf{s})$ we do* not *have (s,a) exits T and (ii) for each $\mathsf{s},\mathsf{s}' \in T$ there is a finite path $\mathsf{w} = \mathsf{s}\mathsf{a}_0 \ldots \mathsf{a}_n\mathsf{s}' \in (T \times B)^* \times T$, i.e. the path stays inside T and only uses actions in B.*

Intuitively, ECs correspond to bottom strongly connected components of the Markov chains induced by possible strategies, so for some pair of strategies all possible paths starting in the EC remain there. An end component T is a *maximal end component (MEC)* if there is no other end component T' such that $T \subseteq T'$. Given an SG G, the set of its MECs is denoted by MEC(G).

Note that, to stay in an EC in an SG, the two players would have to cooperate, since it depends on the pair of strategies. To take into account the adversarial behaviour of the players, it is also relevant to look at a subclass of ECs, the so called *simple end components*, introduced in [KKKW18].

Definition 4 (Simple end component (SEC) [KKKW18]**).** *An EC T is called* simple*, if for all $\mathsf{s} \in T$ it holds that $\mathsf{V}(\mathsf{s}) = \mathsf{bestExit}(T, \mathsf{V})$, where*

$$\mathsf{bestExit}(T, f) := \begin{cases} 1 & \text{if } T \cap \mathsf{Goal} \neq \emptyset \\ \max\limits_{\substack{\mathsf{s} \in T \cap \mathsf{S}_\square \\ (\mathsf{s},\mathsf{a}) \text{ exits } T}} f(\mathsf{s},\mathsf{a}) & \text{else} \end{cases}$$

is called the best exit *(of Maximizer) from T according to the function $f : \mathsf{S} \to \mathbb{R}$. To handle the case that there is no exit of Maximizer in T we set $\max_\emptyset = 0$.*

Intuitively, SECs are ECs where Minimizer does not want to use any of her exits, as all of them have a greater value than the best exit of Maximizer. Assigning any value between those of the best exits of Maximizer and Minimizer to all states in the EC is a solution to the Bellman equations, because both players prefer remaining and getting that value to using their exits [KKKW18, Lemma 1]. However, this is suboptimal for Maximizer, as the goal is not reached if the game remains in the EC forever. Hence we "deflate" the upper bounds of SECs, i.e. reduce them to depend on the best exit of Maximizer. T is called maximal simple end component (MSEC), if there is no SEC T' such that $T \subsetneq T'$. Note that in MDPs, treating all MSECs amounts to treating all MECs.

Algorithm 1. Bounded value iteration algorithm for SG (and MDP)

```
1: procedure BVI(SG G, target set Goal, precision ε > 0)
2:     INITIALIZE_BOUNDS
3:     repeat
4:         X ← SIMULATE until LOOPING or state in Goal is hit
5:         UPDATE(X)                          ▷ Bellman updates or their modification
6:         for T ∈ FIND_MSECs(X) do
7:             DEFLATE(T)                     ▷ Decrease the upper bound of MSECs
8:     until U(s_0) − L(s_0) < ε
```

Algorithm 1 rephrases that of [KKKW18] and describes the general structure of all bounded value iteration algorithms that are relevant for this paper. We discuss it here since all our improvements refer to functions (in capitalized font) in it. In the next section, we design new functions, pinpointing the difference to the other papers. The pseudocode of the functions adapted from the other papers can be found, for the reader's convenience, in [AKW19, Appendix A]. Note that to improve readability, we omit the parameters $\mathsf{G}, \mathsf{Goal}, \mathsf{L}$ and U of the functions in the algorithm.

Bounded Value Iteration: For the standard bounded value iteration algorithm, Line 4 does not run a simulation, but just assigns the whole state space S to X[5]. Then it updates all values according to the Bellman equations.

[5] Since we mainly talk about simulation based algorithms, we included this line to make their structure clearer.

After that it finds all the problematic components, the MSECs, and "deflates" them as described in [KKKW18], i.e. it reduces their values to ensure the convergence to the least fixpoint. This suffices for the bounds to converge and the algorithm to terminate [KKKW18, Theorem 2].

Asynchronous Bounded Value Iteration: To tackle the state space explosion problem, *asynchronous* simulation/learning-based algorithms have been developed [MLG05,BCC+14,KKKW18]. The idea is not to update and deflate all states at once, since there might be too many, or since we only have limited information. Instead of considering the whole state space, a path through the SG is sampled by picking in every state one of the actions that look optimal according to the current over-/under-approximation and then sampling a successor of that action. This is repeated until either a target is found, or until the simulation is looping in an EC; the latter case occurs if the heuristic that picks the actions generates a pair of strategies under which both players only pick staying actions in an EC. After the simulation, only the bounds of the states on the path are updated and deflated. Since we pick actions which look optimal in the simulation, we almost surely find an ϵ-optimal strategy and the algorithm terminates [BCC+14, Theorem 3].

3 Algorithm

3.1 Model-Based

Given only limited information, updating cannot be done using $\mathbb{T}$, since the true probabilities are not known. The approach of [BCC+14] is to sample for a high number of steps and accumulate the observed lower and upper bounds on the true value function for each state-action pair. When the number of samples is large enough, the average of the accumulator is used as the new estimate for the state-action pair, and thus the approximations can be improved and the results back-propagated, while giving statistical guarantees that each update was correct. However, this approach has several drawbacks, the biggest of which is that the number of steps before an update can occur is infeasibly large, often larger than the age of the universe, see Table 1 in Sect. 4.

Our improvements to make the algorithm practically usable are linked to constructing a partial model of the given system. That way, we have more information available on which we can base our estimates, and we can be less conservative when giving bounds on the possible errors. The shift from model-free to model-based learning asymptotically increases the memory requirements from $\mathcal{O}(|\mathsf{S}| \cdot |\mathsf{A}|)$ (as in [SLW+06,BCC+14]) to $\mathcal{O}(|\mathsf{S}|^2 \cdot |\mathsf{A}|)$. However, for systems where each action has a small constant bound on the number of successors, which is typical for many practical systems, e.g. classical PRISM benchmarks, it is still $\mathcal{O}(|\mathsf{S}| \cdot |\mathsf{A}|)$ with a negligible constant difference.

We thus track the number of times some successor t has been observed when playing action a from state s in a variable $\#(\mathsf{s},\mathsf{a},\mathsf{t})$. This implicitly induces the number of times each state-action pair (s,a) has been played $\#(\mathsf{s},\mathsf{a}) =$

$\sum_{\mathsf{t}\in\mathsf{S}} \#(\mathsf{s},\mathsf{a},\mathsf{t})$. Given these numbers we can then calculate probability estimates for every transition as described in the next subsection. They also induce the set of all states visited so far, allowing us to construct a partial model of the game. See [AKW19, Appendix A.2] for the pseudo-code of how to count the occurrences during the simulations.

3.2 Safe Updates with Confidence Intervals Using Distributed Error Probability

We use the counters to compute a lower estimate of the transition probability for some error tolerance $\delta_{\mathbb{T}}$ as follows: We view sampling t from state-action pair (s,a) as a Bernoulli sequence, with success probability $\mathbb{T}(\mathsf{s},\mathsf{a},\mathsf{t})$, the number of trials $\#(\mathsf{s},\mathsf{a})$ and the number of successes $\#(\mathsf{s},\mathsf{a},\mathsf{t})$. The tightest lower estimate we can give using the Hoeffding bound (see [AKW19, Appendix D.1]) is

$$\widehat{\mathbb{T}}(\mathsf{s},\mathsf{a},\mathsf{t}) := \max(0, \frac{\#(s,a,t)}{\#(s,a)} - c), \tag{1}$$

where the confidence width $c := \sqrt{\frac{\ln(\delta_{\mathbb{T}})}{-2\#(\mathsf{s},\mathsf{a})}}$. Since c could be greater than 1, we limit the lower estimate to be at least 0. Now we can give modified update equations:

$$\widehat{\mathsf{L}}(\mathsf{s},\mathsf{a}) := \sum_{\mathsf{t}:\#(\mathsf{s},\mathsf{a},\mathsf{t})>0} \widehat{\mathbb{T}}(\mathsf{s},\mathsf{a},\mathsf{t}) \cdot \mathsf{L}(\mathsf{t})$$

$$\widehat{\mathsf{U}}(\mathsf{s},\mathsf{a}) := \left(\sum_{\mathsf{t}:\#(\mathsf{s},\mathsf{a},\mathsf{t})>0} \widehat{\mathbb{T}}(\mathsf{s},\mathsf{a},\mathsf{t}) \cdot \mathsf{U}(\mathsf{t})\right) + \left(1 - \sum_{\mathsf{t}:\#(\mathsf{s},\mathsf{a},\mathsf{t})>0} \widehat{\mathbb{T}}(\mathsf{s},\mathsf{a},\mathsf{t})\right)$$

The idea is the same for both upper and lower bound: In contrast to the usual Bellman equation (see Sect. 2.2) we use $\widehat{\mathbb{T}}$ instead of $\mathbb{T}$. But since the sum of all the lower estimates does not add up to one, there is some remaining probability for which we need to under-/over-approximate the value it can achieve. We use

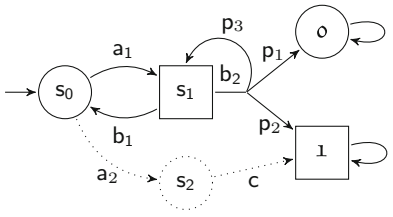


Fig. 1. A running example of an SG. The dashed part is only relevant for the later examples. For actions with only one successor, we do not depict the transition probability 1 (e.g. $\mathbb{T}(\mathsf{s}_0,\mathsf{a}_1,\mathsf{s}_1)$). For state-action pair $(\mathsf{s}_1,\mathsf{b}_2)$, the transition probabilities are parameterized and instantiated in the examples where they are used.

the safe approximations 0 and 1 for the lower and upper bound respectively; this is why in $\widehat{\mathsf{L}}$ there is no second term and in $\widehat{\mathsf{U}}$ the whole remaining probability is added. Algorithm 2 shows the modified update that uses the lower estimates; the proof of its correctness is in [AKW19, Appendix D.2].

Lemma 1 (UPDATE is correct). *Given correct under- and over-approximations* L, U *of the value function* V*, and correct lower probability estimates* $\widehat{\mathbb{T}}$*, the under- and over-approximations after an application of* UPDATE *are also correct.*

Algorithm 2. New update procedure using the probability estimates

1: **procedure** UPDATE(State set X)
2: **for** $f \in \{\mathsf{L}, \mathsf{U}\}$ **do** ▷ For both functions
3: **for** $\mathsf{s} \in X \setminus \mathsf{Goal}$ **do** ▷ For all non-target states in the given set
4: $$f(\mathsf{s}) = \begin{cases} \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \widehat{f}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in \mathsf{S}_{\square} \\ \min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \widehat{f}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in \mathsf{S}_{\bigcirc} \end{cases}$$

Example 1. We illustrate how the calculation works and its huge advantage over the approach from [BCC+14] on the SG from Fig. 1. For this example, ignore the dashed part and let $\mathsf{p}_1 = \mathsf{p}_2 = 0.5$, i.e. we have no self loop, and an even chance to go to the target $\mathtt{1}$ or a sink $\mathtt{o}$. Observe that hence $\mathsf{V}(\mathsf{s}_0) = \mathsf{V}(\mathsf{s}_1) = 0.5$.

Given an error tolerance of $\delta = 0.1$, the algorithm of [BCC+14] would have to sample for more than 10^9 steps before it could attempt a single update. In contrast, assume we have seen 5 samples of action b_2, where 1 of them went to $\mathtt{1}$ and 4 of them to $\mathtt{o}$. Note that, in a sense, we were unlucky here, as the observed averages are very different from the actual distribution. The confidence width for $\delta_{\mathbb{T}} = 0.1$ and 5 samples is $\sqrt{\ln(0.1)/-2 \cdot 5} \approx 0.48$. So given that data, we get $\widehat{\mathbb{T}}(\mathsf{s}_1, \mathsf{b}_2, \mathtt{1}) = \max(0, 0.2 - 0.48) = 0$ and $\widehat{\mathbb{T}}(\mathsf{s}_1, \mathsf{b}_2, \mathtt{o}) = \max(0, 0.8 - 0.48) = 0.32$. Note that both probabilities are in fact lower estimates for their true counterpart.

Assume we already found out that $\mathtt{o}$ is a sink with value 0; how we gain this knowledge is explained in the following subsections. Then, after getting only these 5 samples, UPDATE already decreases the upper bound of $(\mathsf{s}_1, \mathsf{b}_2)$ to 0.68, as we know that at least 0.32 of $\mathbb{T}(\mathsf{s}_1, \mathsf{b}_2)$ goes to the sink.

Given 500 samples of action b_2, the confidence width of the probability estimates already has decreased below 0.05. Then, since we have this confidence width for both the upper and the lower bound, we can decrease the total precision for $(\mathsf{s}_1, \mathsf{b}_2)$ to 0.1, i.e. return an interval in the order of $[0.45; 0.55]$. ◁

Summing up: with the model-based approach we can already start updating after very few steps and get a reasonable level of confidence with a realistic number of samples. In contrast, the state-of-the-art approach of [BCC+14] needs a very large number of samples even for this toy example.

Since for UPDATE we need an error tolerance for every transition, we need to distribute the given total error tolerance δ over all transitions in the current

partial model. For all states in the explored partial model $\widehat{\mathsf{S}}$ we know the number of available actions and can over-approximate the number of successors as $\frac{1}{p_{\min}}$. Thus the error tolerance for each transition can be set to $\delta_{\mathbb{T}} := \frac{\delta \cdot p_{\min}}{|\{\mathsf{a} | \mathsf{s} \in \widehat{\mathsf{S}} \wedge \mathsf{a} \in \mathsf{Av}(\mathsf{s})\}|}$. This is illustrated in Example 4 in [AKW19, Appendix B].

Note that the fact that the error tolerance $\delta_{\mathbb{T}}$ for every transition is the same does *not* imply that the confidence width for every transition is the same, as the latter becomes smaller with increasing number of samples $\#(\mathsf{s}, \mathsf{a})$.

3.3 Improved EC Detection

As mentioned in the description of Algorithm 1, we must detect when the simulation is stuck in a bottom EC and looping forever. However, we may also stop simulations that are looping in some EC but still have a possibility to leave it; for a discussion of different heuristics from [BCC+14,KKKW18], see [AKW19, Appendix A.3].

We choose to define LOOPING as follows: Given a candidate for a bottom EC, we continue sampling until we are $\delta_{\mathbb{T}}$-*sure* (i.e. the error probability is smaller than $\delta_{\mathbb{T}}$) that we cannot leave it. Then we can safely deflate the EC, i.e. decrease all upper bounds to zero.

To detect that something is a $\delta_{\mathbb{T}}$-*sure* EC, we do not sample for the astronomical number of steps as in [BCC+14], but rather extend the approach to detect bottom strongly connected components from [DHKP16]. If in the EC-candidate T there was some state-action pair (s, a) that actually has a probability to exit the T, that probability is at least $p_{\min}$. So after sampling (s, a) for n times, the probability to overlook such a leaving transition is $(1 - p_{\min})^n$ and it should be smaller than $\delta_{\mathbb{T}}$. Solving the inequation for the required number of samples n yields $n \geq \frac{\ln(\delta_{\mathbb{T}})}{\ln(1-p_{min})}$.

Algorithm 3 checks that we have seen all staying state-action pairs n times, and hence that we are $\delta_{\mathbb{T}}$-*sure* that T is an EC. Note that we restrict to staying state-action pairs, since the requirement for an EC is only that there exist staying actions, not that all actions stay. We further speed up the EC-detection, because we do not wait for n samples in every simulation, but we use the aggregated counters that are kept over all simulations.

Algorithm 3. Check whether we are $\delta_{\mathbb{T}}$-*sure* that T is an EC

```
1: procedure δ_T-sure EC (State set T)
2:     requiredSamples = ln(δ_T) / ln(1−p_min)
3:     B ← {(s, a) | s ∈ T ∧ ¬(s, a) exits T}        ▷ Set of staying state-action pairs
4:     return ⋀_{(s,a)∈B} #(s, a) > requiredSamples
```

We stop a simulation, if LOOPING returns true, i.e. under the following three conditions: (i) We have seen the current state before in this simulation ($\mathsf{s} \in X$),

i.e. there is a cycle. (ii) This cycle is explainable by an EC T in our current partial model. (iii) We are $\delta_{\mathbb{T}}$-*sure* that T is an EC.

Algorithm 4. Check if we are probably looping and should stop the simulation

```
1: procedure LOOPING(State set X, state s)
2:     if s ∉ X then
3:         return false                          ▷ Easy improvement to avoid overhead
4:     return ∃T ⊆ X.T is EC in partial model ∧ s ∈ T ∧ δ_T-sure EC(T)
```

Example 2. For this example, we again use the SG from Fig. 1 without the dashed part, but this time with $\mathsf{p}_1 = \mathsf{p}_2 = \mathsf{p}_3 = \frac{1}{3}$. Assume the path we simulated is $(\mathsf{s}_0, \mathsf{a}_1, \mathsf{s}_1, \mathsf{b}_2, \mathsf{s}_1)$, i.e. we sampled the self-loop of action b_2. Then $\{\mathsf{s}_1\}$ is a candidate for an EC, because given our current observation it seems possible that we will continue looping there forever. However, we do not stop the simulation here, because we are not yet $\delta_{\mathbb{T}}$-*sure* about this. Given $\delta_{\mathbb{T}} = 0.1$, the required samples for that are 6, since $\frac{\ln(0.1)}{\ln(1-\frac{1}{3})} = 5.6$. With high probability (greater than $(1 - \delta_{\mathbb{T}}) = 0.9$), within these 6 steps we will sample one of the other successors of $(\mathsf{s}_1, \mathsf{b}_2)$ and thus realise that we should not stop the simulation in s_1. If, on the other hand, we are in state o or if in state s_1 the guiding heuristic only picks b_1, then we are in fact looping for more than 6 steps, and hence we stop the simulation. ◁

3.4 Adapting to Games: Deflating MSECs

To extend the algorithm of [BCC+14] to SGs, instead of collapsing problematic ECs we deflate them as in [KKKW18], i.e. given an MSEC, we reduce the upper bound of all states in it to the upper bound of the $\mathsf{bestExit}$ of Maximizer. In contrast to [KKKW18], we cannot use the upper bound of the $\mathsf{bestExit}$ based on the true probability, but only based on our estimates. Algorithm 5 shows how to deflate an MSEC and highlights the difference, namely that we use $\widehat{\mathsf{U}}$ instead of U.

Algorithm 5. Black box algorithm to deflate a set of states

```
1: procedure DEFLATE(State set X)
2:     for s ∈ X do
3:         U(s) = min(U(s), bestExit(X, Û))
```

The remaining question is how to find MSECs. The approach of [KKKW18] is to find MSECs by removing the suboptimal actions of Minimizer according to the current lower bound. Since it converges to the true value function, all

MSECs are eventually found [KKKW18, Lemma 2]. Since Algorithm 6 can only access the SG as a black box, there are two differences: We can only compare our estimates of the lower bound $\widehat{\mathsf{L}}(\mathsf{s},\mathsf{a})$ to find out which actions are suboptimal. Additionally there is the problem that we might overlook an exit from an EC, and hence deflate to some value that is too small; thus we need to check that any state set FIND_MSECs returns is a $\delta_{\mathbb{T}}$-*sure* EC. This is illustrated in Example 3. For a bigger example of how all our functions work together, see Example 5 in [AKW19, Appendix B].

Algorithm 6. Finding MSECs in the game restricted to X for black box setting

1: **procedure** FIND_MSECs(State set X)
2: $\quad suboptAct_{\bigcirc} \leftarrow \{(\mathsf{s}, \{\mathsf{a} \in \mathsf{Av}(\mathsf{s}) \mid \widehat{\mathsf{L}}(\mathsf{s},\mathsf{a}) > \mathsf{L}(\mathsf{s})\} \mid \mathsf{s} \in \mathsf{S}_{\bigcirc} \cap X\}$
3: $\quad \mathsf{Av}' \leftarrow \mathsf{Av}$ without $suboptAct_{\bigcirc}$
4: $\quad \mathsf{G}' \leftarrow \mathsf{G}$ restricted to states X and available actions Av'
5: $\quad$ **return** $\{T \in \mathsf{MEC}(\mathsf{G}') \mid \delta_{\mathbb{T}}\text{-}sure\ \mathsf{EC}(T)\}$

Example 3. For this example, we use the full SG from Fig. 1, including the dashed part, with $\mathsf{p}_1, \mathsf{p}_2 > 0$. Let $(\mathsf{s}_0, \mathsf{a}_1, \mathsf{s}_1, \mathsf{b}_2, \mathsf{s}_2, \mathsf{b}_1, \mathsf{s}_1, \mathsf{a}_2, \mathsf{s}_2, \mathsf{c}, \mathsf{1})$ be the path generated by our simulation. Then in our partial view of the model, it seems as if $T = \{\mathsf{s}_0, \mathsf{s}_1\}$ is an MSEC, since using a_2 is suboptimal for the minimizing state s_0[6] and according to our current knowledge $\mathsf{a}_1, \mathsf{b}_1$ and b_2 all stay inside T. If we deflated T now, all states would get an upper bound of 0, which would be incorrect.

Thus in Algorithm 6 we need to require that T is an EC $\delta_{\mathbb{T}}$-*surely*. This was not satisfied in the example, as the state-action pairs have not been observed the required number of times. Thus we do not deflate T, and our upper bounds stay correct. Having seen $(\mathsf{s}_1, \mathsf{b}_2)$ the required number of times, we probably know that it is exiting T and hence will not make the mistake. ◁

3.5 Guidance and Statistical Guarantee

It is difficult to give statistical guarantees for the algorithm we have developed so far (i.e. Algorithm 1 calling the new functions from Sects. 3.2, 3.3 and 3.4). Although we can bound the error of each function, applying them repeatedly can add up the error. Algorithm 7 shows our approach to get statistical guarantees: It interleaves a guided simulation phase (Lines 7–10) with a guaranteed standard bounded value iteration (called BVI phase) that uses our new functions (Lines 11–16).

The simulation phase builds the partial model by exploring states and remembering the counters. In the first iteration of the main loop, it chooses actions randomly. In all further iterations, it is guided by the bounds that the last BVI

[6] For $\delta_{\mathbb{T}} = 0.2$, sampling the path to target once suffices to realize that $\mathsf{L}(\mathsf{s}_0, \mathsf{a}_2) > 0$.

phase computed. After $\mathcal{N}_k$ simulations (see below for a discussion of how to choose $\mathcal{N}_k$), all the gathered information is used to compute one version of the partial model with probability estimates $\widehat{\mathbb{T}}$ for a certain error tolerance δ_k. We can continue with the assumption, that these probability estimates are correct, since it is only violated with a probability smaller than our error tolerance (see below for an explanation of the choice of δ_k). So in our correct partial model, we re-initialize the lower and upper bound (Line 12), and execute a guaranteed standard BVI. If the simulation phase already gathered enough data, i.e. explored the relevant states and sampled the relevant transitions often enough, this BVI achieves a precision smaller than ε in the initial state, and the algorithm terminates. Otherwise we start another simulation phase that is guided by the improved bounds.

Algorithm 7. Full algorithm for black box setting

```
1: procedure BlackVI(SG G, target set Goal, precision ε > 0, error tolerance δ > 0)
2:     INITIALIZE_BOUNDS
3:     k = 1                                          ▷ guaranteed BVI counter
4:     Ŝ ← ∅                                          ▷ current partial model
5:     repeat
6:         k ← 2 · k
7:         δ_k ← δ/k
       // Guided simulation phase
8:         for N_k times do
9:             X ← SIMULATE
10:            Ŝ ← Ŝ ∪ X
       // Guaranteed BVI phase
11:        δ_T ← δ_k·p_min / |{a | s∈Ŝ ∧ a∈Av(s)}|      ▷ Set δ_T as described in Section 3.2
12:        INITIALIZE_BOUNDS
13:        for k · |Ŝ| times do
14:            UPDATE(Ŝ)
15:            for T ∈ FIND_MSECs(Ŝ) do
16:                DEFLATE(T)
17:    until U(s_0) − L(s_0) < ε
```

Choice of δ_k: For each of the full BVI phases, we construct a partial model that is correct with probability $(1 - \delta_k)$. To ensure that the sum of these errors is not larger than the specified error tolerance δ, we use the variable k, which is initialised to 1 and doubled in every iteration of the main loop. Hence for the i-th BVI, $k = 2^i$. By setting $\delta_k = \frac{\delta}{k}$, we get that $\sum_{i=1}^{\infty} \delta_k = \sum_{i=1}^{\infty} \frac{\delta}{2^i} = \delta$, and hence the error of all BVI phases does not exceed the specified error tolerance.

When to Stop Each BVI-Phase: The BVI phase might not converge if the probability estimates are not good enough. We increase the number of iterations for each BVI depending on k, because that way we ensure that it eventually is allowed to run long enough to converge. On the other hand, since we always run for finitely many iterations, we also ensure that, if we do not have enough information yet, BVI is eventually stopped. Other stopping criteria could return arbitrarily imprecise results [HM17]. We also multiply with $\left|\widehat{\mathsf{S}}\right|$ to improve the chances of the early BVIs to converge, as that number of iterations ensures that every value has been propagated through the whole model at least once.

Discussion of the Choice of $\mathcal{N}_k$: The number of simulations between the guaranteed BVI phases can be chosen freely; it can be a constant number every time, or any sequence of natural numbers, possibly parameterised by e.g. k, $\left|\widehat{\mathsf{S}}\right|$, ε or any of the parameters of G. The design of particularly efficient choices or learning mechanisms that adjust them on the fly is an interesting task left for future work. We conjecture the answer depends on the given SG and "task" that the user has for the algorithm: E.g. if one just needs a quick general estimate of the behaviour of the model, a smaller choice of $\mathcal{N}_k$ is sensible; if on the other hand a definite precision ε certainly needs to be achieved, a larger choice of $\mathcal{N}_k$ is required.

Theorem 1. *For any choice of sequence for $\mathcal{N}_k$, Algorithm 7 is an anytime algorithm with the following property: When it is stopped, it returns an interval for $\mathsf{V}(\mathsf{s}_0)$ that is PAC[7] for the given error tolerance δ and some ε', with $0 \leq \varepsilon' \leq 1$.*

Theorem 1 is the foundation of the practical usability of our algorithm. Given some time frame and some $\mathcal{N}_k$, it calculates an approximation for $\mathsf{V}(\mathsf{s}_0)$ that is probably correct. Note that the precision ε' is independent of the input parameter ε, and could in the worst case be always 1. However, practically it often is good (i.e. close to 0) as seen in the results in Sect. 4. Moreover, in our modified algorithm, we can also give a convergence guarantee as in [BCC+14]. Although mostly out of theoretical interest, in [AKW19, Appendix D.4] we design such a sequence $\mathcal{N}_k$, too. Since this a-priori sequence has to work in the worst case, it depends on an infeasibly large number of simulations.

Theorem 2. *There exists a choice of $\mathcal{N}_k$, such that Algorithm 7 is PAC for any input parameters ε, δ, i.e. it terminates almost surely and returns an interval for $\mathsf{V}(\mathsf{s}_0)$ of width smaller than ε that is correct with probability at least $1 - \delta$.*

[7] Probably Approximately Correct, i.e. with probability greater than $1 - \delta$, the value lies in the returned interval of width ε'.

3.6 Utilizing the Additional Information of Grey Box Input

In this section, we consider the grey box setting, i.e. for every state-action pair (s, a) we additionally know the exact number of successors $|\mathsf{Post}(\mathsf{s}, \mathsf{a})|$. Then we can sample every state-action pair until we have seen all successors, and hence this information amounts to having qualitative information about the transitions, i.e. knowing where the transitions go, but not with which probability.

In that setting, we can improve the EC-detection and the estimated bounds in UPDATE. For EC-detection, note that the whole point of $\delta_{\mathbb{T}}$*-sure* EC is to check whether there are further transitions available; in grey box, we know this and need not depend on statistics. For the bounds, note that the equations for $\widehat{\mathsf{L}}$ and $\widehat{\mathsf{U}}$ both have two parts: The usual Bellman part and the remaining probability multiplied with the most conservative guess of the bound, i.e. 0 and 1. If we know all successors of a state-action pair, we do not have to be as conservative; then we can use $\min_{\mathsf{t} \in \mathsf{Post}(\mathsf{s},\mathsf{a})} \mathsf{L}(\mathsf{t})$ respectively $\max_{\mathsf{t} \in \mathsf{Post}(\mathsf{s},\mathsf{a})} \mathsf{U}(\mathsf{t})$. Both these improvements have huge impact, as demonstrated in Sect. 4. However, of course, they also assume more knowledge about the model.

4 Experimental Evaluation

We implemented the approach as an extension of PRISM-Games [CFK+13a]. 11 MDPs with reachability properties were selected from the Quantitative Verification Benchmark Set [HKP+19]. Further, 4 stochastic games benchmarks from [CKJ12, SS12, CFK+13b, CKPS11] were also selected. We ran the experiments on a 40 core Intel Xeon server running at 2.20 GHz per core and having 252 GB of RAM. The tool however utilised only a single core and 1 GB of memory for the model checking. Each benchmark was ran 10 times with a timeout of 30 min. We ran two versions of Algorithm 7, one with the SG as a black box, the other as a grey box (see Definition 2). We chose $\mathcal{N}_k = 10,000$ for all iterations. The tool stopped either when a precision of 10^{-8} was obtained or after 30 min. In total, 16 different model-property combinations were tried out. The results of the experiment are reported in Table 1.

In the black box setting, we obtained $\varepsilon < 0.1$ on 6 of the benchmarks. 5 benchmarks were 'hard' and the algorithm did not improve the precision below 1. For 4 of them, it did not even finish the first simulation phase. If we decrease $\mathcal{N}_k$, the BVI phase is entered, but still no progress is made.

In the grey box setting, on 14 of 16 benchmarks, it took only 6 min to achieve $\varepsilon < 0.1$. For 8 these, the exact value was found within that time. Less than 50% of the state space was explored in the case of `pacman`, `pneuli-zuck-3`, `rabin-3`, `zeroconf` and `cloud_5`. A precision of $\varepsilon < 0.01$ was achieved on 15/16 benchmarks over a period of 30 min.

Table 1. Achieved precision ε' given by our algorithm in both grey and black box settings after running for a period of 30 min (See the paragraph below Theorem 1 for why we use ε' and not ε). The first set of the models are MDPs and the second set are SGs. '-' indicates that the algorithm did not finish the first simulation phase and hence partial BVI was not called. m is the number of steps required by the DQL algorithm of [BCC+14] before the first update. As this number is very large, we report only $log_{10}(m)$. For comparison, note that the age of the universe is approximately 10^{26} ns; logarithm of number of steps doable in this time is thus in the order of 26.

Model	States	Explored %	Precision		$log_{10}(m)$
		Grey/Black	Grey	Black	
consensus	272	100/100	0.00945	0.171	338
csma-2-2	1,038	93/93	0.00127	0.2851	1,888
firewire	83,153	55/-	0.0057	1	129,430
ij-3	7	100/100	0	0.0017	2,675
ij-10	1,023	100/100	0	0.5407	17
pacman	498	18/47	0.00058	0.0086	1,801
philosophers-3	956	56/21	0	1	2,068
pnueli-zuck-3	2,701	25/71	0	0.0285	5,844
rabin-3	27,766	7/4	0	0.026	110,097
wlan-0	2,954	100/100	0	0.8667	9,947
zeroconf	670	29/27	0.00007	0.0586	5,998
cdmsn	1,240	100/98	0	0.8588	3,807
cloud-5	8,842	49/20	0.00031	0.0487	71,484
mdsm-1	62,245	69/-	0.09625	1	182,517
mdsm-2	62,245	72/-	0.00055	1	182,517
team-form-3	12,476	64/-	0	1	54,095

Figure 2 shows the evolution of the lower and upper bounds in both the grey- and the black box settings for 4 different models. Graphs for the other models as well as more details on the results are in [AKW19, Appendix C].

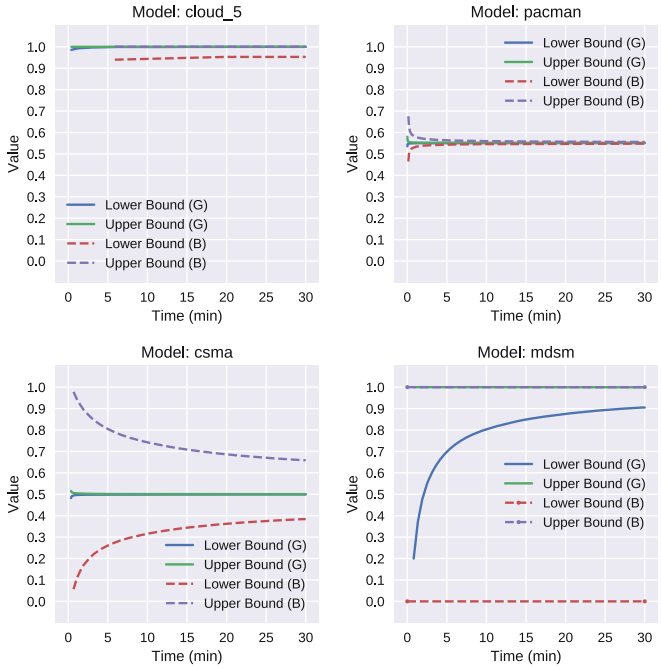


Fig. 2. Performance of our algorithm on various MDP and SG benchmarks in grey and black box settings. Solid lines denote the bounds in the grey box setting while dashed lines denote the bounds in the black box setting. The plotted bounds are obtained after each partial BVI phase, because of which they do not start at $[0, 1]$ and not at time 0. Graphs of the remaining benchmarks may be found in [AKW19, Appendix C].

5 Conclusion

We presented a PAC SMC algorithm for SG (and MDP) with the reachability objective. It is the first one for SG and the first practically applicable one. Nevertheless, there are several possible directions for further improvements. For instance, one can consider different sequences for lengths of the simulation phases, possibly also dependent on the behaviour observed so far. Further, the error tolerance could be distributed in a non-uniform way, allowing for fewer visits in rarely visited parts of end components. Since many systems are strongly connected, but at the same time feature some infrequent behaviour, this is the next bottleneck to be attacked. [KM19]

References

[AKW19] Ashok, P., Křetínský, J.: Maximilian Weininger. PAC statistical model checking for markov decision processes and stochastic games. Technical Report arXiv.org/abs/1905.04403 (2019)

[BBB+10] Basu, A., Bensalem, S., Bozga, M., Caillaud, B., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. In: Hatcliff, J., Zucca, E. (eds.) FMOODS/FORTE 2010. LNCS, vol. 6117, pp. 32–46. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13464-7_4

[BCC+14] Brázdil, T., et al.: Verification of markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_8

[BCLS13] Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: a flexible, distributable statistical model checking library. In: Joshi, K., Siegle, M., Stoelinga, M., DArgenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 160–164. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40196-1_12

[BDL+12] Bulychev, P.E., et al.: UPPAAL-SMC: statistical model checking for priced timed automata. In: QAPL (2012)

[BFHH11] Bogdoll, J., Ferrer Fioriti, L.M., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: Bruni, R., Dingel, J. (eds.) FMOODS/FORTE 2011. LNCS, vol. 6722, pp. 59–74. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21461-5_4

[BHH12] Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and statistical model checking for modestly nondeterministic models. In: Schmitt, J.B. (ed.) MMB&DFT 2012. LNCS, vol. 7201, pp. 249–252. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28540-0_20

[BK08] Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press (2008). ISBN 978-0-262-02649-9

[BT99] Brafman, R.I., Tennenholtz, M.: A near-optimal poly-time algorithm for learning a class of stochastic games. In: IJCAI, pp. 734–739 (1999)

[CFK+13a] Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: a model checker for stochastic multi-player games. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 185–191. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36742-7_13

[CFK+13b] Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Meth. Syst. Des. **43**(1), 61–92 (2013)

[CH08] Chatterjee, K., Henzinger, T.A.: Value iteration. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking. LNCS, vol. 5000, pp. 107–138. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69850-0_7

[CH12] Chatterjee, K., Henzinger, T.A.: A survey of stochastic ω-regular games. J. Comput. Syst. Sci. **78**(2), 394–413 (2012)

[CKJ12] Calinescu, R., Kikuchi, S., Johnson, K.: Compositional reverification of probabilistic safety properties for large-scale complex IT systems. In: Calinescu, R., Garlan, D. (eds.) Monterey Workshop 2012. LNCS, vol. 7539, pp. 303–329. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34059-8_16

[CKPS11] Chen, T., Kwiatkowska, M., Parker, D., Simaitis, A.: Verifying team formation protocols with probabilistic model checking. In: Leite, J., Torroni,

P., Ågotnes, T., Boella, G., van der Torre, L. (eds.) CLIMA 2011. LNCS (LNAI), vol. 6814, pp. 190–207. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22359-4_14

[Con92] Condon, A.: The complexity of stochastic games. Inf. Comput. **96**(2), 203–224 (1992)

[CZ11] Clarke, E.M., Zuliani, P.: Statistical model checking for cyber-physical systems. In: ATVA, pp. 1–12 (2011)

[DDL+12] David, A., et al.: Statistical model checking for stochastic hybrid systems. In: HSB, pp. 122–136 (2012)

[DDL+13] David, A., Du, D., Guldstrand Larsen, K., Legay, A., Mikučionis, M.: Optimizing control strategy using statistical model checking. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 352–367. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_24

[DHKP16] Daca, P., Henzinger, T.A., Křetínský, J., Petrov, T.: Faster statistical model checking for unbounded temporal properties. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 112–129. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_7

[DHS18] D'Argenio, P.R., Hartmanns, A., Sedwards, S.: Lightweight statistical model checking in nondeterministic continuous time. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 336–353. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03421-4_22

[DLL+11a] David, A., et al.: Statistical model checking for networks of priced timed automata. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 80–96. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24310-3_7

[DLL+11b] David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_27

[DLST15] D'Argenio, P., Legay, A., Sedwards, S., Traonouez, L.-M.: Smart sampling for lightweight verification of markov decision processes. STTT **17**(4), 469–484 (2015)

[EGF12] Ellen, C., Gerwinn, S., Fränzle, M.: Confidence bounds for statistical model checking of probabilistic hybrid systems. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 123–138. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33365-1_10

[FT14] Fu, J., Topcu, U.: Probably approximately correct MDP learning and control with temporal logic constraints. In: Robotics: Science and Systems (2014)

[HAK18] Hasanbeig, M., Abate, A., Kroening, D.: Logically-correct reinforcement learning. CoRR, 1801.08099 (2018)

[HAK19] Hasanbeig, M., Abate, A., Kroening, D.: Certified reinforcement learning with logic guidance. CoRR, abs/1902.00778 (2019)

[HJB+10] He, R., Jennings, P., Basu, S., Ghosh, A.P., Wu, H.: A bounded statistical approach for model checking of unbounded until properties. In: ASE, pp. 225–234 (2010)

[HKP+19] Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: TACAS 2019 (2019, to appear)

[HLMP04] Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24622-0_8

[HM17] Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. (2017)

[HMZ+12] Henriques, D., Martins, J., Zuliani, P., Platzer, A., Clarke, E.M.: Statistical model checking for Markov decision processes. In: QEST, pp. 84–93 (2012)

[HPS+19] Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Omega-regular objectives in model-free reinforcement learning. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 395–412. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_27

[JCL+09] Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03845-7_15

[JLS12] Jegourel, C., Legay, A., Sedwards, S.: A platform for high performance statistical model checking – PLASMA. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 498–503. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_37

[KKKW18] Kelmendi, E., Krämer, J., Křetínský, J., Weininger, M.: Value iteration for simple stochastic games: stopping criterion and learning algorithm. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 623–642. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_36

[KM19] Křetínský, J., Meggendorfer, T.: Of cores: a partial-exploration framework for Markov decision processes. Submitted 2019

[KNP11] Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47

[Lar12] Larsen, K.G.: Statistical model checking, refinement checking, optimization,. for stochastic hybrid systems. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 7–10. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33365-1_2

[Lar13] Guldstrand Larsen, K.: Priced timed automata and statistical model checking. In: Johnsen, E.B., Petre, L. (eds.) IFM 2013. LNCS, vol. 7940, pp. 154–161. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38613-8_11

[Lit94] Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: ICML, pp. 157–163 (1994)

[LN81] Lakshmivarahan, S., Narendra, K.S.: Learning algorithms for two-person zero-sum stochastic games with incomplete information. Math. Oper. Res. **6**(3), 379–386 (1981)

[LP08] Lassaigne, R., Peyronnet, S.: Probabilistic verification and approximation. Ann. Pure Appl. Logic **152**(1–3), 122–131 (2008)
[LP12] Lassaigne, R., Peyronnet, S.: Approximate planning and verification for large Markov decision processes. In: SAC, pp. 1314–1319, (2012)
[LST14] Legay, A., Sedwards, S., Traonouez, L.-M.: Scalable verification of markov decision processes. In: Canal, C., Idani, A. (eds.) SEFM 2014. LNCS, vol. 8938, pp. 350–362. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15201-1_23
[Mar75] Martin, D.A.: Borel determinacy. Ann. Math. **102**(2), 363–371 (1975)
[MLG05] Mcmahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: In ICML 2005, pp. 569–576 (2005)
[Nor98] Norris, J.R.: Markov Chains. Cambridge University Press, Cambridge (1998)
[PGL+13] Palaniappan, S.K., Gyori, B.M., Liu, B., Hsu, D., Thiagarajan, P.S.: Statistical model checking based calibration and analysis of bio-pathway models. In: Gupta, A., Henzinger, T.A. (eds.) CMSB 2013. LNCS, vol. 8130, pp. 120–134. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40708-6_10
[Put14] Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, Hoboken (2014)
[RF91] Raghavan, T.E.S., Filar, J.A.: Algorithms for stochastic games – a survey. Z. Oper. Res. **35**(6), 437–472 (1991)
[RP09] El Rabih, D., Pekergin, N.: Statistical model checking using perfect simulation. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 120–134. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04761-9_11
[SB98] Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
[SKC+14] Sadigh, D., Kim, E.S., Coogan, S., Sastry, S.S.S., Sanjit, A.: A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In: CDC, pp. 1091–1096 (2014)
[SLW+06] Strehl, A.L., Li, L., Wiewiora, E., Langford, J., Littman, M.L.: PAC model-free reinforcement learning. In: ICML, pp. 881–888 (2006)
[SS12] Saffre, F., Simaitis, A.: Host selection through collective decision. ACM Trans. Auton. Adapt. Syst. **7**(1), 4:1–4:16 (2012)
[SVA04] Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 202–215. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_16
[SVA05] Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 266–280. Springer, Heidelberg (2005). https://doi.org/10.1007/11513988_26
[WT16] Wen, M., Topcu, U.: Probably approximately correct learning in stochastic games with temporal logic specifications. In: IJCAI, pp. 3630–3636 (2016)
[YCZ10] Younes, H.L.S., Clarke, E.M., Zuliani, P.: Statistical verification of probabilistic properties with unbounded until. In: Davies, J., Silva, L., Simao, A. (eds.) SBMF 2010. LNCS, vol. 6527, pp. 144–160. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19829-8_10

[YKNP06] Younes, H.L.S., Kwiatkowska, M.Z., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. STTT **8**(3), 216–228 (2006)

[YS02a] Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_17

[ZPC10] Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to simulink/stateflow verification. In: HSCC, pp. 243–252 (2010)

Symbolic Monitoring Against Specifications Parametric in Time and Data

Masaki Waga[1,2,3(✉)], Étienne André[1,4,5], and Ichiro Hasuo[1,2]

[1] National Institute of Informatics, Tokyo, Japan
mwaga@nii.ac.jp
[2] SOKENDAI (The Graduate University for Advanced Studies), Tokyo, Japan
[3] JSPS Research Fellow, Tokyo, Japan
[4] Université Paris 13, LIPN, CNRS, UMR 7030, 93430 Villetaneuse, France
[5] JFLI, CNRS, Tokyo, Japan

Abstract. Monitoring consists in deciding whether a log meets a given specification. In this work, we propose an automata-based formalism to monitor logs in the form of actions associated with time stamps and arbitrarily data values over infinite domains. Our formalism uses both timing parameters and data parameters, and is able to output answers symbolic in these parameters and in the log segments where the property is satisfied or violated. We implemented our approach in an ad-hoc prototype SyMon, and experiments show that its high expressive power still allows for efficient online monitoring.

1 Introduction

Monitoring consists in checking whether a sequence of data (a log or a signal) satisfies or violates a specification expressed using some formalism. Offline monitoring consists in performing this analysis after the system execution, as the technique has access to the entire log in order to decide whether the specification is violated. In contrast, online monitoring can make a decision earlier, ideally as soon as a witness of the violation of the specification is encountered.

Using existing formalisms (e.g., the metric first order temporal logic [14]), one can check whether a given bank customer withdraws more than 1,000 € every week. With formalisms extended with data, one may even *identify* such customers. Or, using an extension of the signal temporal logic (STL) [18], one can ask: "is that true that the value of variable x is always copied to y exactly 4 time units later?" However, questions relating time and data using parameters become

This work is partially supported by JST ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), by JSPS Grants-in-Aid No. 15KT0012 & 18J22498 and by the ANR national research program PACS (ANR-14-CE28-0002).

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 520–539, 2019.
https://doi.org/10.1007/978-3-030-25540-4_30

much harder (or even impossible) to express using existing formalisms: "what are the users and time frames during which a user withdraws more than half of the total bank withdrawals within seven days?" And even, can we *synthesize* the durations (not necessarily 7 days) for which this specification holds? Or "what is the set of variables for which there exists a duration within which their value is always copied to another variable?" In addition, detecting periodic behaviors without knowing the period can be hard to achieve using existing formalisms.

In this work, we address the challenging problem to monitor logs enriched with both timing information and (infinite domain) data. In addition, we significantly push the existing limits of expressiveness so as to allow for a further level of abstraction using *parameters*: our specification can be both parametric in the *time* and in the *data*. The answer to this symbolic monitoring is richer than a pure Boolean answer, as it *synthesizes* the values of both time and data parameters for which the specification holds. This allows us notably to detect periodic behaviors without knowing the period while being symbolic in terms of data. For example, we can *synthesize variable names* (data) and *delays* for which variables will have their value copied to another data within the aforementioned delay. In addition, we show that we can detect the log *segments* (start and end date) for which a specification holds.

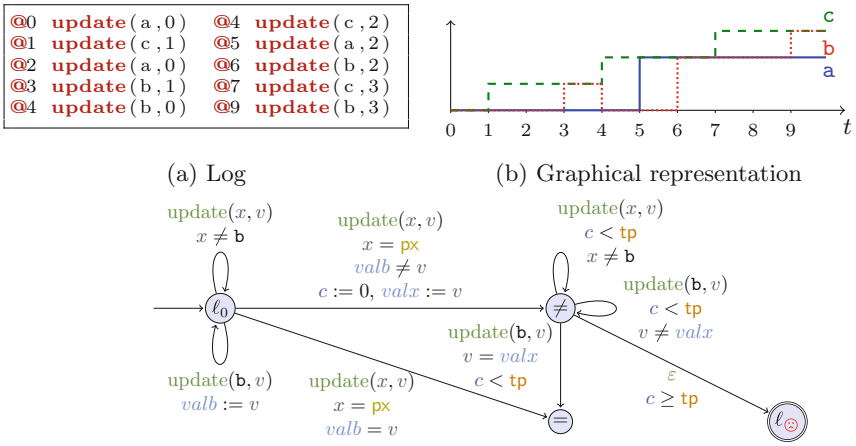
Example 1. Consider a system updating three variables `a`, `b` and `c` (i. e., strings) to values (rationals). An example of log is given in Fig. 1a. Although our work is event-based, we can give a graphical representation similar to that of signals in Fig. 1b. Consider the following property: "for any variable px, whenever an update of that variable occurs, then within strictly less than tp time units, the value of variable `b` must be equal to that update". The *variable parameter* px is compared with string values and the *timing parameter* tp is used in the timing constraints. We are interested in checking for which values of px and tp this property is violated. This can be seen as a synthesis problem in both the variable and timing parameters. For example, $\mathsf{px} = \mathtt{c}$ and $\mathsf{tp} = 1.5$ is a violation of the specification, as the update of `c` to 2 at time 4 is not propagated to `b` within 1.5 time unit. Our algorithm outputs such violation by a constraint e.g., $\mathsf{px} = c \wedge \mathsf{tp} \leq 2$. In contrast, the value of any signal at any time is always such that either `b` is equal to that signal, or the value of `b` will be equal to that value within at most 2 time units. Thus, the specification holds for any valuation of the variable parameter px, provided $\mathsf{tp} > 2$.

We propose an automata-based approach to perform monitoring parametric in both time and data. We implement our work in a prototype SyMon and perform experiments showing that, while our formalism allows for high expressiveness, it is also tractable even for online monitoring.

We believe our framework balances expressiveness and monitoring performance well: *(i)* Regarding expressiveness, comparison with the existing work is summarized in Table 1 (see Sect. 2 for further details). *(ii)* Our monitoring is *complete*, in the sense that it returns a symbolic constraint characterizing *all* the parameter valuations that match a given specification. *(iii)* We also achieve

Table 1. Comparison of monitoring expressiveness

Work	[7]	[18]	[14]	[13]	[30]	[26]	[4]	[9]	This work
Timing parameters	√	×	?	?	?	×	√	×	√
Data	√	√	√	√	√	√	×	√	√
Parametric data	√	×	√	√	√	√	×	√	√
Memory	×	√	√	√	√	√	×	×	√
Aggregation	×	×	×	√	√	×	×	×	√
Complete parameter identification	√	N/A	√/	√/	N/A	N/A	√	√	√



(a) Log (b) Graphical representation

(c) Monitoring PTDA

Fig. 1. Monitoring copy to b within tp time units

reasonable monitoring speed, especially given the degree of parametrization in our formalism. Note that it is not easy to formally claim superiority in expressiveness: proofs would require arguments such as the pumping lemma; and such formal comparison does not seem to be a concern of the existing work. Moreover, such formal comparison bears little importance for industrial practitioners: expressivity via an elaborate encoding is hardly of practical use. We also note that, in the existing work, we often observe gaps between the formalism in a theory and the formalism that the resulting tool actually accepts. This is not the case with the current framework.

Outline. After discussing related works in Sect. 2, we introduce the necessary preliminaries in Sect. 3, and our parametric timed data automata in Sect. 4. We present our symbolic monitoring approach in Sect. 5 and conduct experiments in Sect. 6. We conclude in Sect. 7.

2 Related Works

Robustness and Monitoring. Robust (or quantitative) monitoring extends the binary question whether a log satisfies a specification by asking "by how much" the specification is satisfied. The quantification of the distance between a signal and a signal temporal logic (STL) specification has been addressed in, e.g., [20–23,25,27] (or in a slightly different setting in [5]). The distance can be understood in terms of space ("signals") or time. In [6], the distance also copes for reordering of events. In [10], the *robust pattern matching problem* is considered over signal regular expressions, by quantifying the distance between the signal regular expression specification and the *segments* of the signal. For piecewise-constant and piecewise-linear signals, the problem can be effectively solved using a finite union of convex polyhedra. While our framework does not fit in robust monitoring, we can simulate both the robustness w.r.t. time (using timing parameters) and w.r.t. data, e.g., signal values (using data parameters).

Monitoring with Data. The tool MarQ [30] performs monitoring using Quantified Event Automata (QEA) [12]. This approach and ours share the automata-based framework, the ability to express some first-order properties using "events containing data" (which we encode using local variables associated with actions), and data may be quantified. However, [30] does not seem to natively support specification parametric in time; in addition, [30] does not perform complete ("symbolic") parameters synthesis, but outputs the violating entries of the log.

The metric first order temporal logic (MFOTL) allows for a high expressiveness by allowing universal and existential quantification over data—which can be seen as a way to express parameters. A monitoring algorithm is presented for a safety fragment of MFOTL in [14]. Aggregation operators are added in [13], allowing to compute *sums* or *maximums* over data. A fragment of this logics is implemented in MonPoly [15]. While these works are highly expressive, they do not natively consider timing parameters; in addition, MonPoly does not output symbolic answers, i. e., symbolic conditions on the parameters to ensure validity of the formula.

In [26], binary decision diagrams (BDDs) are used to symbolically represent the observed data in QTL. This can be seen as monitoring data against a parametric specification, with a symbolic internal encoding. However, their implementation DejaVu only outputs *concrete* answers. In contrast, we are able to provide symbolic answers (both in timing and data parameters), e.g., in the form of union of polyhedra for rationals, and unions of string constraints using equalities ($=$) and inequalities ($\neq$).

Freeze Operator. In [18], STL is extended with a freeze operator that can "remember" the value of a signal, to compare it to a later value of the same signal. This logic STL* can express properties such as "In the initial 10 s, x copies the values of y within a delay of 4 s": $\mathbf{G}_{[0,10]} * (\mathbf{G}_{[0,4]} y^* = x)$. While the setting is somehow different (STL* operates over signals while we operate over timed data words), the requirements such as the one above can easily be encoded

in our framework. In addition, we are able to *synthesize* the delay within which the values are always copied, as in Example 1. In contrast, it is not possible to determine using STL* which variables and which delays violate the specification.

Monitoring with Parameters. In [7], a log in the form of a dense-time real-valued signal is tested against a parameterized extension of STL, where parameters can be used to model uncertainty both in signal values and in timing values. The output comes in the form of a subset of the parameters space for which the formula holds on the log. In [9], the focus is only on signal parameters, with an improved efficiency by reusing techniques from the *robust* monitoring. Whereas [7,9] fit in the framework of signals and temporal logics while we fit in words and automata, our work shares similarities with [7,9] in the sense that we can express data parameters; in addition, [9] is able as in our work to exhibit the segment of the log associated with the parameters valuations for which the specification holds. A main difference however is that we can use memory and aggregation, thanks to arithmetic on variables.

In [24], the problem of *inferring* temporal logic formulae with constraints that hold in a given numerical data time series is addressed.

Timed Pattern Matching. A recent line of work is that of timed pattern matching, that takes as input a log and a specification, and decides *where* in the log the specification is satisfied or violated. On the one hand, a line of works considers signals, with specifications either in the form of timed regular expressions [11,31–33], or a temporal logic [34]. On the other hand, a line of works considers timed words, with specifications in the form of timed automata [4,36]. We will see that our work can also encode parametric timed pattern matching. Therefore, our work can be seen as a two-dimensional extension of both lines of works: first, we add timing parameters ([4] also considers similar timing parameters) and, second, we add data—themselves extended with parameters. That is, coming back to Example 1, [31–33,36] could only infer the segments of the log for which the property is violated for a given (fixed) variable and a given (fixed) timing parameter; while [4] could infer both the segments of the log and the timing parameter valuations, but not which variable violates the specification.

Summary. We compare related works in Table 1. "Timing parameters" denote the ability to synthesize unknown constants used in timing constraints (e.g., modalities intervals, or clock constraints). "**?**" denotes works not natively supporting this, although it might be encoded. The term "Data" refers to the ability to manage logs over infinite domains (apart from timestamps). For example, the log in Fig. 1a features, beyond timestamps, both string (variable name) and rationals (value). Also, works based on real-valued signals are naturally able to manage (at least one type of) data. "Parametric data" refer to the ability to express formulas where data (including signal values) are compared to (quantified or unquantified) variables or unknown parameters; for example, in the log in Fig. 1a, an example of property parametric in data is to synthesize the parameters for which the difference of values between two consecutive updates of

variable px is always below pv, where px is a string parameter and pv a rational-valued parameter. "Memory" is the ability to remember *past* data; this can be achieved using e.g., the freeze operator of STL*, or variables (e.g., in [14,26,30]). "Aggregation" is the ability to aggregate data using operators such as sum or maximum; this allows to express properties such as "A user must not withdraw more than $10,000 within a 31 day period" [13]. This can be supported using dedicated aggregation operators [13] or using variables ([30], and our work). "Complete parameter identification" denotes the *synthesis* of the set of parameters that satisfy or violate the property. Here, "N/A" denotes the absence of parameter [18], or when parameters are used in a way (existentially or universally quantified) such as the identification is not explicit (instead, the position of the log where the property is violated is returned [26]). In contrast, we return in a *symbolic* manner (as in [4,7]) the exact set of (data and timing) parameters for which a property is satisfied. "√//×" denotes "yes" in the theory paper, but not in the tool.

3 Preliminaries

Clocks, Timing Parameters and Timed Guards. We assume a set $\mathbb{C} = \{c_1, \ldots, c_H\}$ of *clocks*, i. e., real-valued variables that evolve at the same rate. A *clock valuation* is $\nu : \mathbb{C} \to \mathbb{R}_{\geq 0}$. We write $\mathbf{0}$ for the clock valuation assigning 0 to all clocks. Given $d \in \mathbb{R}_{\geq 0}$, $\nu + d$ is s.t. $(\nu + d)(c) = \nu(c) + d$, for all $c \in \mathbb{C}$. Given $R \subseteq \mathbb{C}$, we define the *reset* of a valuation ν, denoted by $[\nu]_R$, as follows: $[\nu]_R(c) = 0$ if $c \in R$, and $[\nu]_R(c) = \nu(c)$ otherwise.

We assume a set $\mathbb{TP} = \{\mathsf{tp}_1, \ldots, \mathsf{tp}_J\}$ of *timing parameters*. A *timing parameter valuation* is $\gamma : \mathbb{TP} \to \mathbb{Q}_+$. We assume $\bowtie \in \{<, \leq, =, \geq, >\}$. A *timed guard* tg is a constraint over $\mathbb{C} \cup \mathbb{TP}$ defined by a conjunction of inequalities of the form $c \bowtie d$, or $c \bowtie \mathsf{tp}$ with $d \in \mathbb{N}$ and $\mathsf{tp} \in \mathbb{TP}$. Given tg, we write $\nu \models \gamma(tg)$ if the expression obtained by replacing each c with $\nu(c)$ and each tp with $\gamma(\mathsf{tp})$ in tg evaluates to true.

Variables, Data Parameters and Data Guards. For sake of simplicity, we assume a *single* infinite domain $\mathbb{D}$ for data. The formalism defined in Sect. 4 can be extended in a straightforward manner to different domains for different variables (and our implementation does allow for different types). The case of *finite* data domain is immediate too. We define this formalism in an *abstract* manner, so as to allow a sort of parameterized domain.

We assume a set $\mathbb{V} = \{v_1, \ldots, v_M\}$ of *variables* valued over $\mathbb{D}$. These variables are internal variables, that allow an high expressive power in our framework, as they can be compared or updated to other variables or parameters. We also assume a set $\mathbb{LV} = \{lv_1, \ldots, lv_O\}$ of *local variables* valued over $\mathbb{D}$. These variables will only be used locally along a transition in the "argument" of the action (e.g., x and v in upate(x, v)), and in the associated guard and (right-hand part of) updates. We assume a set $\mathbb{VP} = \{\mathsf{vp}_1, \ldots, \mathsf{vp}_N\}$ of *data parameters*, i. e., unknown variable constants.

A *data type* $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ is made of *(i)* an infinite domain $\mathbb{D}$, *(ii)* a set of admissible Boolean expressions $\mathcal{DE}$ (that may rely on $\mathbb{V}$, $\mathbb{LV}$ and $\mathbb{VP}$), which will define the type of guards over variables in our subsequent automata, and *(iii)* a domain for updates $\mathcal{DU}$ (that may rely on $\mathbb{V}$, $\mathbb{LV}$ and $\mathbb{VP}$), which will define the type of updates of variables in our subsequent automata.

Example 2. As a first example, let us define the data type for rationals. We have $\mathbb{D} = \mathbb{Q}$. Let us define Boolean expressions. A *rational comparison* is a constraint over $\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$ defined by a conjunction of inequalities of the form $v \bowtie d$, $v \bowtie v'$, or $v \bowtie \mathsf{vp}$ with $v, v' \in \mathbb{V} \cup \mathbb{LV}$, $d \in \mathbb{Q}$ and $\mathsf{vp} \in \mathbb{VP}$. $\mathcal{DE}$ is the set of all rational comparisons over $\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$. Let us then define updates. First, a linear arithmetic expression over $\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$ is $\sum_i \alpha_i v_i + \beta$, where $v_i \in \mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$ and $\alpha_i, \beta \in \mathbb{Q}$. Let $\mathcal{LA}(\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP})$ denote the set of arithmetic expressions over $\mathbb{V}$, $\mathbb{LV}$ and $\mathbb{VP}$. We then have $\mathcal{DU} = \mathcal{LA}(\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP})$.

As a second example, let us define the data type for strings. We have $\mathbb{D} = \mathbb{S}$, where $\mathbb{S}$ denotes the set of all strings. A *string comparison* is a constraint over $\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$ defined by a conjunction of comparisons of the form $v \approx s$, $v \approx v'$, or $v \approx \mathsf{vp}$ with $v, v' \in \mathbb{V} \cup \mathbb{LV}$, $s \in \mathbb{S}$, $\mathsf{vp} \in \mathbb{VP}$ and $\approx \in \{=, \neq\}$. $\mathcal{DE}$ is the set of all string comparisons over $\mathbb{V} \cup \mathbb{LV} \cup \mathbb{VP}$. $\mathcal{DU} = \mathbb{V} \cup \mathbb{LV} \cup \mathbb{S}$, i. e., a string variable can be assigned another string variable, or a concrete string.

A *variable valuation* is $\mu : \mathbb{V} \to \mathbb{D}$. A *local variable valuation* is a partial function $\eta : \mathbb{LV} \nrightarrow \mathbb{D}$. A *data parameter valuation* is $\zeta : \mathbb{VP} \to \mathbb{D}$. Given a data guard $dg \in \mathcal{DE}$, a variable valuation μ, a local variable valuation η defined for the local variables in dg, and a data parameter valuation ζ, we write $(\mu, \eta) \models \zeta(dg)$ if the expression obtained by replacing within dg all occurrences of each data parameter vp_i by $\zeta(\mathsf{vp}_i)$ and all occurrences of each variable v_j (resp. local variable lv_k) with its concrete valuation $\mu(v_j)$ (resp. $\eta(lv_k)$)) evaluates to true.

A parametric data update is a partial function $\mathsf{PDU} : \mathbb{V} \nrightarrow \mathcal{DU}$. That is, we can assign to a variable an expression over data parameters and other variables, according to the data type. Given a parametric data update PDU, a variable valuation μ, a local variable valuation η (defined for all local variables appearing in PDU), and a data parameter valuation ζ, we define $[\mu]_{\eta(\zeta(\mathsf{PDU}))} : \mathbb{V} \to \mathbb{D}$ as:

$$[\mu]_{\eta(\zeta(\mathsf{PDU}))}(v) = \begin{cases} \mu(v) & \text{if } \mathsf{PDU}(v) \text{ is undefined} \\ \eta(\mu(\zeta(\mathsf{PDU}(v)))) & \text{otherwise} \end{cases}$$

where $\eta(\mu(\zeta(\mathsf{PDU}(v))))$ denotes the replacement within the update expression $\mathsf{PDU}(v)$ of all occurrences of each data parameter vp_i by $\zeta(\mathsf{vp}_i)$, and all occur-

Table 2. Variables, parameters and valuations used in guards

	Timed guards		Data guards		
	Clock	Timing parameter	(Data) variable	Local variable	Data parameter
Variable	c	tp	v	lv	vp
Valuation	ν	γ	μ	η	ζ

rences of each variable v_j (resp. local variable lv_k) with its concrete valuation $\mu(v_j)$ (resp. $\eta(lv_k)$). Observe that this replacement gives a value in $\mathbb{D}$, therefore the result of $[\mu]_{\eta(\zeta(\mathsf{PDU}))}$ is indeed a data parameter valuation $\mathbb{V} \to \mathbb{D}$. That is, $[\mu]_{\eta(\zeta(\mathsf{PDU}))}$ computes the new (non-parametric) variable valuation obtained after applying to μ the partial function PDU valuated with ζ.

Example 3. Consider the data type for rationals, the variables set $\{v_1, v_2\}$, the local variables set $\{lv_1, lv_2\}$ and the parameters set $\{\mathsf{vp}_1\}$. Let μ be the variable valuation such that $\mu(v_1) = 1$ and $\mu(v_2) = 2$, and η be the local variable valuation such that $\eta(lv_1) = 2$ and $\eta(lv_2)$ is not defined. Let ζ be the data parameter valuation such that $\zeta(\mathsf{vp}_1) = 1$. Consider the parametric data update function PDU such that $\mathsf{PDU}(v_1) = 2 \times v_1 + v_2 - lv_1 + \mathsf{vp}_1$, and $\mathsf{PDU}(v_2)$ is undefined. Then the result of $[\mu]_{\eta(\zeta(\mathsf{PDU}))}$ is μ' such that $\mu'(v_1) = 2 \times \mu(v_1) + \mu(v_2) - \eta(lv_1) + \zeta(\mathsf{vp}_1) = 3$ and $\mu'(v_2) = 2$.

4 Parametric Timed Data Automata

We introduce here Parametric timed data automata (PTDAs). They can be seen as an extension of parametric timed automata [2] (that extend timed automata [1] with parameters in place of integer constants) with unbounded data variables and parametric variables. PTDAs can also be seen as an extension of some extensions of timed automata with data (see e.g., [16,19,29]), that we again extend with both data parameters and timing parameters. Or as an extension of quantified event automata [12] with explicit time representation using clocks, and further augmented with timing parameters. PTDAs feature both timed guards and data guards; we summarize the various variables and parameters types together with their notations in Table 2.

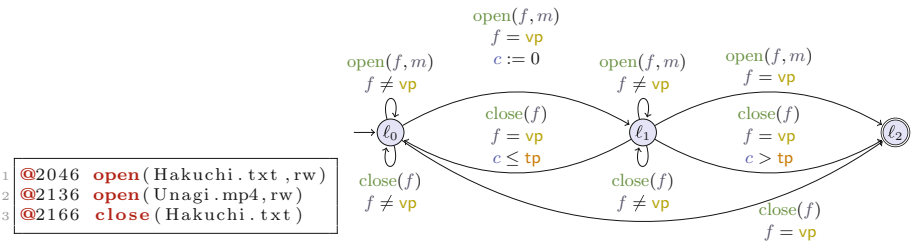
We will associate local variables with actions (which can be see as *predicates*). Let $Dom : \Sigma \to 2^{\mathbb{LV}}$ denote the set of local variables associated with each action. Let $Var(dg)$ (resp. $Var(\mathsf{PDU})$) denote the set of variables occurring in dg (resp. PDU).

Definition 1 (PTDA). *Given a data type $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$, a parametric timed data automaton (PTDA) $\mathcal{A}$ over this data type is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, F, \mathbb{C}, \mathbb{TP}, \mathbb{V}, \mathbb{LV}, \mu_0, \mathbb{VP}, E)$, where:*

1. *Σ is a finite set of actions,*
2. *L is a finite set of locations, $\ell_0 \in L$ is the initial location,*
3. *$F \subseteq L$ is the set of accepting locations,*
4. *$\mathbb{C}$ is a finite set of clocks,*
5. *$\mathbb{TP}$ is a finite set of timing parameters,*
6. *$\mathbb{V}$ (resp. $\mathbb{LV}$) is a finite set of variables (resp. local variables) over $\mathbb{D}$,*
7. *μ_0 is the initial variable valuation,*
8. *$\mathbb{VP}$ is a finite set of data parameters,*

9. *E is a finite set of edges* $e = (\ell, tg, dg, a, R, \mathsf{PDU}, \ell')$ *where (i)* $\ell, \ell' \in L$ *are the source and target locations, (ii)* tg *is a timed guard, (iii)* $dg \in \mathcal{DE}$ *is a data guard such as* $Var(dg) \cap \mathbb{LV} \subseteq Dom(a)$*, (iv)* $a \in \Sigma$*, (v)* $R \subseteq \mathbb{C}$ *is a set of clocks to be reset, and (vi)* $\mathsf{PDU} : \mathbb{V} \nrightarrow \mathcal{DU}$ *is the parametric data update function such that* $Var(\mathsf{PDU}) \cap \mathbb{LV} \subseteq Dom(a)$*.*

The domain conditions on dg and PDU ensure that the local variables used in the guard (resp. update) are only those in the action signature $Dom(a)$.



(a) Example of log (b) PTDA monitor

Fig. 2. Monitoring proper file opening and closing

Example 4. Consider the PTDA in Fig. 2b over the data type for strings. We have $\mathbb{C} = \{c\}$, $\mathbb{TP} = \{\mathsf{tp}\}$, $\mathbb{V} = \emptyset$ and $\mathbb{LV} = \{f, m\}$. $Dom(\text{open}) = \{f, m\}$ while $Dom(\text{close}) = \{f\}$. ℓ_2 is the only accepting location, modeling the violation of the specification.

This PTDA (freely inspired by a formula from [26] further extended with timing parameters) monitors the improper file opening and closing, i. e., a file already open should not be open again, and a file that is open should not be closed too late. The data parameter vp is used to *symbolically* monitor a given file name, i. e., we are interested in opening and closings of this file only, while other files are disregarded (specified using the self-loops in ℓ_0 and ℓ_1 with data guard $f \neq \mathsf{vp}$). Whenever f is opened (transition from ℓ_0 to ℓ_1), a clock c is reset. Then, in ℓ_1, if f is closed within tp time units (timed guard "$c \leq \mathsf{tp}$"), then the system goes back to ℓ_0. However, if instead f is opened again, this is an incorrect behavior and the system enters ℓ_2 via the upper transition. The same occurs if f is closed more than tp time units after opening.

Given a data parameter valuation ζ and a timing parameter valuation γ, we denote by $\gamma|\zeta(\mathcal{A})$ the resulting *timed data automaton (TDA)*, i. e., the non-parametric structure where all occurrences of a parameter vp_i (resp. tp_j) have been replaced by $\zeta(\mathsf{vp}_i)$ (resp. $\gamma(\mathsf{tp}_j)$). Note that, if $\mathbb{V} = \mathbb{LV} = \emptyset$, then $\mathcal{A}$ is a *parametric timed automaton* [2] and $\gamma|\zeta(\mathcal{A})$ is a *timed automaton* [1].

We now equip our TDAs with a concrete semantics.

Definition 2 (Semantics of a TDA). *Given a PTDA* $\mathcal{A} = (\Sigma, L, \ell_0, F, \mathbb{C}, \mathbb{TP}, \mathbb{V}, \mathbb{LV}, \mu_0, \mathbb{VP}, E)$ *over a data type* $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$*, a data parameter valuation* ζ *and a timing parameter valuation* γ*, the semantics of* $\gamma|\zeta(\mathcal{A})$ *is given by the timed transition system (TTS)* $(S, s_0, \rightarrow)$*, with*

- $S = L \times \mathbb{D}^M \times \mathbb{R}^H_{\geq 0}$, $s_0 = (\ell_0, \mu_0, \mathbf{0})$,
- $\rightarrow$ *consists of the discrete and (continuous) delay transition relations:*

1. *discrete transitions:* $(\ell, \mu, \nu) \overset{e,\eta}{\mapsto} (\ell', \mu', \nu')$*, there exist* $e = (\ell, tg, dg, a, R, \mathsf{PDU}, \ell') \in E$ *and a local variable valuation* η *defined exactly for* $Dom(a)$*, such that* $\nu \models \gamma(tg)$*,* $(\mu, \eta) \models \zeta(dg)$*,* $\nu' = [\nu]_R$*, and* $\mu' = [\mu]_{\eta(\zeta(\mathsf{PDU}))}$.
2. *delay transitions:* $(\ell, \mu, \nu) \overset{d}{\mapsto} (\ell, \mu, \nu + d)$*, with* $d \in \mathbb{R}_{\geq 0}$.

Moreover we write $((\ell, \mu, \nu), (e, \eta, d), (\ell', \mu', \nu')) \in \rightarrow$ for a combination of a delay and discrete transition if $\exists \nu'' : (\ell, \mu, \nu) \overset{d}{\mapsto} (\ell, \mu, \nu'') \overset{e,\eta}{\mapsto} (\ell', \mu', \nu')$.

Given a TDA $\gamma|\zeta(\mathcal{A})$ with concrete semantics $(S, s_0, \rightarrow)$, we refer to the states of S as the *concrete states* of $\gamma|\zeta(\mathcal{A})$. A *run* of $\gamma|\zeta(\mathcal{A})$ is an alternating sequence of concrete states of $\gamma|\zeta(\mathcal{A})$ and triples of edges, local variable valuations and delays, starting from the initial state s_0 of the form $(\ell_0, \mu_0, \nu_0), (e_0, \eta, d_0), (\ell_1, \mu_1, \nu_1), \cdots$ with $i = 0, 1, \ldots$, $e_i \in E$, $d_i \in \mathbb{R}_{\geq 0}$ and $((\ell_i, \mu_i, \nu_i), (e_i, \eta_i, d_i), (\ell_{i+1}, \mu_{i+1}, \nu_{i+1})) \in \rightarrow$. Given such a run, the associated *timed data word* is $(a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \cdots$, where a_i is the action of edge e_{i-1}, η_i is the local variable valuation associated with that transition, and $\tau_i = \sum_{0 \leq j \leq i-1} d_j$, for $i = 1, 2 \cdots$. For a timed data word w and a concrete state (ℓ, μ, ν) of $\gamma|\zeta(\mathcal{A})$, we write $(\ell_0, \mu_0, \mathbf{0}) \overset{w}{\rightarrow} (\ell, \mu, \nu)$ in $\gamma|\zeta(\mathcal{A})$ if w is associated with a run of $\gamma|\zeta(\mathcal{A})$ of the form $(\ell_0, \mu_0, \mathbf{0}), \ldots, (\ell_n, \mu_n, \nu_n)$ with $(\ell_n, \mu_n, \nu_n) = (\ell, \mu, \nu)$. For a timed data word $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots, (a_n, \tau_n, \eta_n)$, we denote $|w| = n$ and for any $i \in \{1, 2, \ldots, n\}$, we denote $w(1, i) = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots, (a_i, \tau_i, \eta_i)$.

A finite run is *accepting* if its last state (ℓ, μ, ν) is such that $\ell \in F$. The *language* $\mathcal{L}(\gamma|\zeta(\mathcal{A}))$ is defined to be the set of timed data words associated with all accepting runs of $\gamma|\zeta(\mathcal{A})$.

Example 5. Consider the PTDA in Fig. 2b over the data type for strings. Let $\gamma(\mathtt{tp}) = 100$ and $\zeta(\mathtt{vp}) = \mathtt{Hakuchi.txt}$. An accepting run of the TDA $\gamma|\zeta(\mathcal{A})$ is: $(\ell_0, \emptyset, \nu_0), (e_0, \eta_0, 2046), (\ell_1, \emptyset, \nu_1), (e_1, \eta_1, 90), (\ell_1, \emptyset, \nu_2)(e_2, \eta_2, 30), (\ell_2, \emptyset, \nu_3)$, where $\emptyset$ denotes a variable valuation over an empty domain (recall that $\mathbb{V} = \emptyset$ in Fig. 2b), $\nu_0(c) = 0$, $\nu_1(c) = 0$, $\nu_2(c) = 90$, $\nu_3(c) = 120$, e_0 is the upper edge from ℓ_0 to ℓ_1, e_1 is the self-loop above ℓ_1, e_2 is the lower edge from ℓ_1 to ℓ_2, $\eta_0(f) = \eta_2(f) = \mathtt{Hakuchi.txt}$, $\eta_1(f) = \mathtt{Unagi.mp4}$, $\eta_0(m) = \eta_1(m) = \mathtt{rw}$, and $\eta_2(m)$ is undefined (because $Dom(\mathrm{close}) = \{f\}$).

The associated timed data word is $(\mathrm{open}, 2046, \eta_0), (\mathrm{open}, 2136, \eta_1), (\mathrm{close}, 2166, \eta_2)$.

Since each action is associated with a set of local variables, given an ordering on this set, it is possible to see a given action and a variable valuation as a predicate: for example, assuming an ordering of $\mathbb{LV}$ such as f precedes m, then open

with η_0 can be represented as open(Hakuchi.txt, rw). Using this convention, the log in Fig. 2a corresponds exactly to this timed data word.

5 Symbolic Monitoring Against PTDA Specifications

In symbolic monitoring, in addition to the (observable) actions in Σ, we employ *unobservable* actions denoted by ε and satisfying $Dom(\varepsilon) = \emptyset$. We write Σ_ε for $\Sigma \sqcup \{\varepsilon\}$. We let η_ε be the local variable valuation such that $\eta_\varepsilon(lv)$ is undefined for any $lv \in \mathbb{LV}$. For a timed data word $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots, (a_n, \tau_n, \eta_n)$ over Σ_ε, the projection $w{\downarrow_\Sigma}$ is the timed data word over Σ obtained from w by removing any triple (a_i, τ_i, η_i) where $a_i = \varepsilon$. An edge $e = (\ell, tg, dg, a, R, \mathsf{PDU}, \ell') \in E$ is *unobservable* if $a = \varepsilon$, and *observable* otherwise. The use of unobservable actions allows us to encode parametric timed pattern matching (see Sect. 5.3).

We make the following assumption on the PTDAs in symbolic monitoring.

Assumption 1. *The PTDA $\mathcal{A}$ does not contain any loop of unobservable edges.*

5.1 Problem Definition

Roughly speaking, given a PTDA $\mathcal{A}$ and a timed data word w, the symbolic monitoring problem asks for the set of pairs $(\gamma, \zeta) \in (\mathbb{Q}_+)^{\mathbb{TP}} \times \mathbb{D}^{\mathbb{VP}}$ satisfying $w(1, i) \in \gamma|\zeta(\mathcal{A})$, where $w(1, i)$ is a prefix of w. Since $\mathcal{A}$ also contains unobservable edges, we consider w' which is w augmented by unobservable actions.

Symbolic monitoring problem:
INPUT: a PTDA $\mathcal{A}$ over a data type $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ and actions Σ_ε, and a timed data word w over Σ
PROBLEM: compute all the pairs (γ, ζ) of timing and data parameter valuations such that there is a timed data word w' over Σ_ε and $i \in \{1, 2, \ldots, |w'|\}$ satisfying $w'{\downarrow_\Sigma} = w$ and $w'(1, i) \in \mathcal{L}(\gamma|\zeta(\mathcal{A}))$. That is, it requires the *validity domain* $D(w, \mathcal{A}) = \{(\gamma, \zeta) \mid \exists w' : i \in \{1, 2, \ldots, |w'|\}, w'{\downarrow_\Sigma} = w \text{ and } w'(1, i) \in \mathcal{L}(\gamma|\zeta(\mathcal{A}))\}$.

Example 6. Consider the PTDA $\mathcal{A}$ and the timed data word w shown in Fig. 1. The validity domain $D(w, \mathcal{A})$ is $D(w, \mathcal{A}) = D_1 \cup D_2$, where

$$D_1 = \{(\gamma, \zeta) \mid 0 \le \gamma(\mathsf{tp}) \le 2, \zeta(\mathsf{xp}) = \mathrm{c}\} \text{ and } D_2 = \{(\gamma, \zeta) \mid 0 \le \gamma(\mathsf{tp}) \le 1, \zeta(\mathsf{xp}) = \mathrm{a}\}.$$

For $w' = w(1, 3) \cdot (\varepsilon, \eta_\varepsilon, 2.9)$, we have $w' \in \mathcal{L}(\gamma|\zeta(\mathcal{A}))$ and $w'{\downarrow_\Sigma} = w(1, 3)$, where γ and ζ are such that $\gamma(\mathsf{tp}) = 1.8$ and $\zeta(\mathsf{xp}) = \mathrm{c}$, and $w(1, 3) \cdot (\varepsilon, \eta_\varepsilon, 2.9)$ denotes the juxtaposition.

For the data types in Example 2, the validity domain $D(w, \mathcal{A})$ can be represented by a constraint of finite size because the length $|w|$ of the timed data word is finite.

5.2 Online Algorithm

Our algorithm is *online* in the sense that it outputs $(\gamma, \zeta) \in D(w, \mathcal{A})$ as soon as its membership is witnessed, even before reading the whole timed data word w.

Let $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots (a_n, \tau_n, \eta_n)$ and $\mathcal{A}$ be the timed data word and PTDA given in symbolic monitoring, respectively. Intuitively, after reading (a_i, τ_i, η_i), our algorithm symbolically computes for all parameter valuations $(\gamma, \zeta) \in (\mathbb{Q}_+)^{\mathbb{TP}} \times \mathbb{D}^{\mathbb{VP}}$ the concrete states (ℓ, ν, μ) satisfying $(\ell_0, \mu_0, \mathbf{0}) \xrightarrow{w(1,i)} (\ell, \mu, \nu)$ in $\gamma|\zeta(\mathcal{A})$. Since $\mathcal{A}$ has unobservable edges as well as observable edges, we have to add unobservable actions before or after observable actions in w. By $Conf_i^o$, we denote the configurations after reading (a_i, τ_i, η_i) and no unobservable actions are appended after (a_i, τ_i, η_i). By $Conf_i^u$, we denote the configurations after reading (a_i, τ_i, η_i) and at least one unobservable action is appended after (a_i, τ_i, η_i).

Definition 3 ($Conf_i^o, Conf_i^u$). *For a PTDA $\mathcal{A}$ over actions Σ_ε, a timed data word w over Σ, and $i \in \{0, 1, \ldots, |w|\}$ (resp. $i \in \{-1, 0, \ldots, |w|\}$), $Conf_i^o$ (resp. $Conf_i^u$) is the set of 5-tuples $(\ell, \nu, \gamma, \mu, \zeta)$ such that there is a timed data word w' over Σ_ε satisfying the following: (i) $(\ell_0, \mu_0, \mathbf{0}) \xrightarrow{w'} (\ell, \mu, \nu)$ in $\gamma|\zeta(\mathcal{A})$, (ii) $w'{\downarrow_\Sigma} = w(1, i)$, (iii) The last action $a'_{|w'|}$ of w' is observable (resp. unobservable and its timestamp is less than τ_{i+1}).*

Algorithm 1. Outline of our algorithm for symbolic monitoring

Input: A PTDA $\mathcal{A} = (\Sigma_\varepsilon, L, \ell_0, F, \mathbb{C}, \mathbb{TP}, \mathbb{V}, \mathbb{LV}, \mu_0, \mathbb{VP}, E)$ over a data type $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ and actions Σ_ε, and a timed data word $w = (a_1, \tau_1, \eta_1), (a_2, \tau_2, \eta_2), \ldots, (a_n, \tau_n, \eta_n)$ over Σ

Output: $\bigcup_{i \in \{1,2,\ldots,n+1\}} Result_i$ is the validity domain $D(w, \mathcal{A})$

1 $Conf_{-1}^u \leftarrow \emptyset$; $Conf_0^o \leftarrow \{(\ell_0, \mathbf{0}, \gamma, \mu_0, \zeta) \mid \gamma \in (\mathbb{Q}_+)^{\mathbb{TP}}, \zeta \in \mathbb{D}^{\mathbb{VP}}\}$
2 **for** $i \leftarrow 1$ **to** n **do**
3 | **compute** $(Conf_{i-1}^u, Conf_i^o)$ **from** $(Conf_{i-2}^u, Conf_{i-1}^o)$
4 | $Result_i \leftarrow \{(\gamma, \zeta) \mid \exists (\ell, \nu, \gamma, \mu, \zeta) \in Conf_{i-1}^u \cup Conf_i^o. \ell \in F\}$
5 **compute** $Conf_n^u$ **from** $(Conf_{n-1}^u, Conf_n^o)$
6 $Result_{n+1} \leftarrow \{(\gamma, \zeta) \mid \exists (\ell, \nu, \gamma, \mu, \zeta) \in Conf_n^u. \ell \in F\}$

Algorithm 1 shows an outline of our algorithm for symbolic monitoring (see [35] for the full version). Our algorithm incrementally computes $Conf_{i-1}^u$ and $Conf_i^o$ (line 3). After reading (a_i, τ_i, η_i), our algorithm stores the partial results $(\gamma, \zeta) \in D(w, \mathcal{A})$ witnessed from the accepting configurations in $Conf_{i-1}^u$ and $Conf_i^o$ (line 4). (We also need to try to take potential unobservable transitions and store the results from the accepting configurations *after* the last element of the timed data word (lines 5 and 6).)

Since $(\mathbb{Q}_+)^{\mathbb{TP}} \times \mathbb{D}^{\mathbb{VP}}$ is an infinite set, we cannot try each $(\gamma, \zeta) \in (\mathbb{Q}_+)^{\mathbb{TP}} \times \mathbb{D}^{\mathbb{VP}}$ and we use a symbolic representation for parameter valuations. Similarly to the

reachability synthesis of parametric timed automata [28], a set of clock and timing parameter valuations can be represented by a convex polyhedron. For variable valuations and data parameter valuations, we need an appropriate representation depending on the data type $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$. Moreover, for the termination of Algorithm 1, some operations on the symbolic representation are required.

Theorem 1 (termination). *For any PTDA $\mathcal{A}$ over a data type $(\mathbb{D}, \mathcal{DE}, \mathcal{DU})$ and actions Σ_ε, and for any timed data word w over Σ, Algorithm 1 terminates if the following operations on the symbolic representation V_d of a set of variable and data parameter valuations terminate.*

1. *restriction and update $\{([\mu]_{\eta(\zeta(\mathsf{PDU}))}, \zeta) \mid \exists (\mu, \zeta) \in V_d.\, (\mu, \eta) \models \zeta(dg)\}$, where η is a local variable valuation, PDU is a parametric data update function, and dg is a data guard;*
2. *emptiness checking of V_d;*
3. *projection $V_d{\downarrow_{\mathbb{VP}}}$ of V_d to the data parameters $\mathbb{VP}$.* □

Example 7. For the data type for rationals in Example 2, variable and data parameter valuations V_d can be represented by convex polyhedra and the above operations terminate. For the data type for strings $\mathbb{S}$ in Example 2, variable and data parameter valuations V_d can be represented by $\mathbb{S}^{|\mathbb{V}|} \times (\mathbb{S} \cup \mathcal{P}_{\text{fin}}(\mathbb{S}))^{|\mathbb{VP}|}$ and the above operations terminate, where $\mathcal{P}_{\text{fin}}(\mathbb{S})$ is the set of finite sets of $\mathbb{S}$.

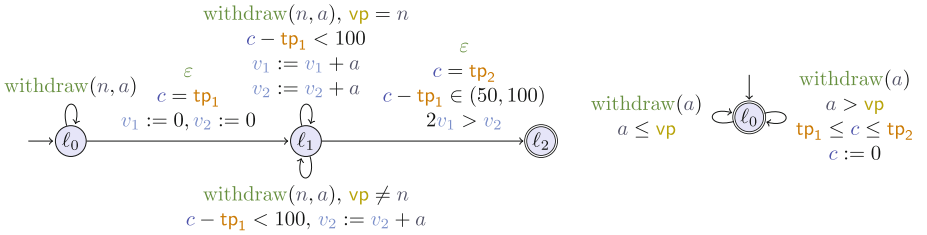


Fig. 3. PTDAs in Dominant (left) and Periodic (right)

5.3 Encoding Parametric Timed Pattern Matching

The symbolic monitoring problem is a generalization of the parametric timed pattern matching problem of [4]. Recall that parametric timed pattern matching aims at synthesizing timing parameter valuations and *start and end times in the log* for which a log segment satisfies or violates a specification. In our approach, by adding a clock measuring the absolute time, and two timing parameters encoding respectively the start and end date of the segment, one can easily infer the log segments for which the property is satisfied.

Consider the Dominant PTDA (left of Fig. 3). It is inspired by a monitoring of withdrawals from bank accounts of various users [15]. This PTDA monitors situations when a user withdraws more than half of the total withdrawals within a time window of $(50, 100)$. The actions are $\Sigma = \{\text{withdraw}\}$

and $Dom(\text{withdraw}) = \{n, a\}$, where n has a string value and a has an integer value. The string n represents a user name and the integer a represents the amount of the withdrawal by the user n. Observe that clock c is never reset, and therefore measures absolute time. The automaton can non-deterministically remain in ℓ_0, or start to measure a log by taking the ε-transition to ℓ_1 checking $c = \mathsf{tp}_1$, and therefore "remembering" the start time using timing parameter tp_1. Then, whenever a user vp has withdrawn more than half of the accumulated withdrawals (data guard $2v_1 > v_2$) in a $(50, 100)$ time window (timed guard $c - \mathsf{tp}_1 \in (50, 100)$), the automaton takes a ε-transition to the accepting location, checking $c = \mathsf{tp}_2$, and therefore remembering the end time using timing parameter tp_2.

6 Experiments

We implemented our symbolic monitoring algorithm in a tool SyMon in C++, where the domain for data is the strings and the integers. Our tool SyMon is distributed at https://github.com/MasWag/symon. We use PPL [8] for the symbolic representation of the valuations. We note that we employ an optimization to merge adjacent polyhedra in the configurations if possible. We evaluated our monitor algorithm against three original benchmarks: Copy in Fig. 1c; and Dominant and Periodic in Fig. 3. We conducted experiments on an Amazon EC2 c4.large instance (2.9 GHz Intel Xeon E5-2666 v3, 2 vCPUs, and 3.75 GiB RAM) that runs Ubuntu 18.04 LTS (64 bit).

6.1 Benchmark 1: Copy

Our first benchmark Copy is a monitoring of variable updates much like the scenario in [18]. The actions are $\Sigma = \{\text{update}\}$ and $Dom(\text{update}) = \{n, v\}$, where n has a string value representing the name of the updated variables and v has an integer value representing the updated value. Our set consists of 10 timed data words of length 4,000 to 40,000.

The PTDA in Copy is shown in Fig. 1c, where we give an additional constraint $3 < \mathsf{tp} < 10$ on tp. The property encoded in Fig. 1c is "for any variable px, whenever an update of that variable occurs, then within tp time units, the value of b must be equal to that update".

The experiment result is in Fig. 4. We observe that the execution time is linear to the number of the events and the memory usage is more or less constant with respect to the number of events.

6.2 Benchmark 2: Dominant

Our second benchmark is Dominant (Fig. 3 left). Our set consists of 10 timed data words of length 2,000 to 20,000. The experiment result is in Fig. 5. We observe that the execution time is linear to the number of the events and the memory usage is more or less constant with respect to the number of events.

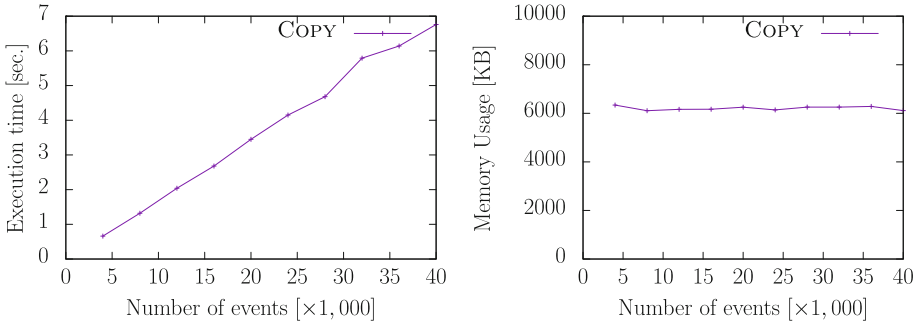


Fig. 4. Execution time (left) and memory usage (right) of Copy

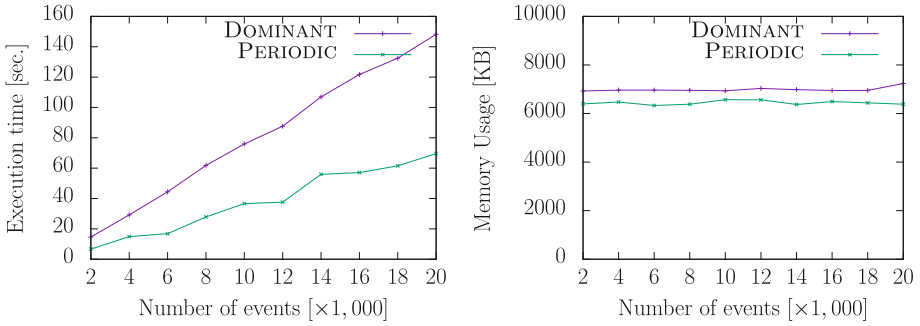


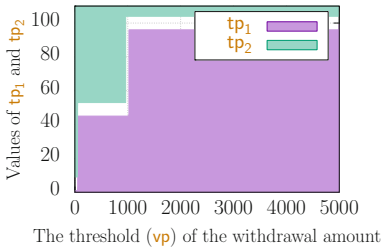
Fig. 5. Execution time (left) and memory usage (right) of Dominant and Periodic

6.3 Benchmark 3: Periodic

Our third benchmark Periodic is inspired by a parameter identification of periodic withdrawals from one bank account. The actions are $\Sigma = \{\text{withdraw}\}$ and $Dom(\text{withdraw}) = \{a\}$, where a has an integer value representing the amount of the withdrawal. We randomly generated a set consisting of 10 timed data words of length 2,000 to 20,000. Each timed data word consists of the following three kinds of periodic withdrawals:

shortperiod One withdrawal occurs every 5 ± 1 time units. The amount of the withdrawal is 50 ± 3.
middleperiod One withdrawal occurs every 50 ± 3 time units. The amount of the withdrawal is 1000 ± 40.
longperiod One withdrawal occurs every 100 ± 5 time units. The amount of the withdrawal is 5000 ± 20.

The PTDA in Periodic is shown in the right of Fig. 3. The PTDA matches situations where, for any two successive withdrawals of amount more than vp, the duration between them is within $[\mathsf{tp}_1, \mathsf{tp}_2]$. By the symbolic monitoring, one can identify the period of the



periodic withdrawals of amount greater than vp is in $[\mathsf{tp}_1, \mathsf{tp}_2]$. An example of the validity domain is shown in the right figure.

The experiment result is in Fig. 5. We observe that the execution time is linear to the number of the events and the memory usage is more or less constant with respect to the number of events.

6.4 Discussion

First, a positive result is that our algorithm effectively performs symbolic monitoring on more than 10,000 actions in one or two minutes even though the PTDAs feature both timing and data parameters. The execution time in COPY is 50–100 times smaller than that in DOMINANT and PERIODIC. This is because the constraint $3 < \mathsf{tp} < 10$ in COPY is strict and the size of the configurations (i. e., $Conf_i^o$ and $Conf_i^u$ in Algorithm 1) is small. Another positive result is that in all of the benchmarks, the execution time is linear and the memory usage is more or less constant in the size of the input word. This is because the size of configurations (i. e., $Conf_i^o$ and $Conf_i^u$ in Algorithm 1) is bounded due to the following reason. In DOMINANT, the loop in ℓ_1 of the PTDA is deterministic, and because of the guard $c - \mathsf{tp}_1 \in (50, 100)$ in the edge from ℓ_1 to ℓ_2, the number of the loop edges at ℓ_1 in an accepting run is bounded (if the duration between two continuing actions are bounded as in the current setting). Therefore, $|Conf_i^o|$ and $|Conf_i^u|$ in Algorithm 1 are bounded. The reason is similar in COPY, too. In PERIODIC, since the PTDA is deterministic and the valuations of the amount of the withdrawals are in finite number, $|Conf_i^o|$ and $|Conf_i^u|$ in Algorithm 1 are bounded.

It is clear that we can design ad-hoc automata for which the execution time of symbolic monitoring can grow much faster (e.g., exponential in the size of input word). However, experiments showed that our algorithm monitors various interesting properties in a reasonable time.

COPY and DOMINANT use data and timing parameters as well as memory and aggregation; from Table 1, no other monitoring tool can compute the valuations satisfying the specification. We however used the parametric timed model checker IMITATOR [3] to try to perform such a synthesis, by encoding the input log as a separate automaton; but IMITATOR ran out of memory (on a 3.75 GiB RAM computer) for DOMINANT with $|w| = 2000$, while SYMON terminates in 14 s with only 6.9 MiB for the same benchmark. Concerning PERIODIC, the only existing work that can possibly accommodate this specification is [7]. While the precise performance comparison is interesting future work (their implementation is not publicly available), we do not expect our implementation be vastly outperformed: in [7], their tool times out (after 10 min) for a simple specification ("$\mathbf{E}_{[0,s_2]}\mathbf{G}_{[0,s_1]}(x < p)$") and a signal discretized by only 128 points.

For those problem instances which MONPOLY and DEJAVU can accommodate (which are simpler and less parametrized than our benchmarks), they tend to run much faster than ours. For example, in [26], it is reported that they can process a trace of length 1,100,004 in 30.3 s. The trade-off here is expressivity: for

example, DejaVu does not seem to accommodate Dominant, because DejaVu does not allow for aggregation. We also note that, while SyMon can be slower than MonPoly and DejaVu, it is fast enough for many scenarios of real-world online monitoring.

7 Conclusion and Perspectives

We proposed a symbolic framework for monitoring using parameters both in data and time. Logs can use timestamps and infinite domain data, while our monitor automata can use timing and variable parameters (in addition to clocks and local variables). In addition, our online algorithm can answer symbolically, by outputting all valuations (and possibly log segments) for which the specification is satisfied or violated. We implemented our approach into a prototype SyMon and experiments showed that our tool can effectively monitor logs of dozens of thousands of events in a short time.

Perspectives. Combining the BDDs used in [26] with some of our data types (typically strings) could improve our approach by making it even more symbolic. Also, taking advantage of the polarity of some parameters (typically the timing parameters, in the line of [17]) could improve further the efficiency.

We considered *infinite* domains, but the case of *finite* domains raises interesting questions concerning result representation: if the answer to a property is "neither `a` nor `b`", knowing the domain is $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, then the answer should be `c`.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
2. Alur, R., Henzinger, T.A., Vardi, M.Y.: Parametric real-time reasoning. In: Kosaraju, S.R., Johnson, D.S., Aggarwal, A. (eds.) STOC, pp. 592–601. ACM, New York (1993). https://doi.org/10.1145/167088.167242
3. André, É., Fribourg, L., Kühne, U., Soulat, R.: IMITATOR 2.5: a tool for analyzing robustness in scheduling problems. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 33–36. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32759-9_6
4. André, É., Hasuo, I., Waga, M.: Offline timed pattern matching under uncertainty. In: Lin, A.W., Sun, J. (eds.) ICECCS, pp. 10–20. IEEE CPS (2018). https://doi.org/10.1109/ICECCS2018.2018.00010
5. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
6. Asarin, E., Basset, N., Degorre, A.: Distance on timed words and applications. In: Jansen, D.N., Prabhakar, P. (eds.) FORMATS 2018. LNCS, vol. 11022, pp. 199–214. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00151-3_12

7. Asarin, E., Donzé, A., Maler, O., Nickovic, D.: Parametric identification of temporal properties. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 147–160. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_12
8. Bagnara, R., Hill, P.M., Zaffanella, E.: The parma polyhedra library: toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Sci. Comput. Program. **72**(1–2), 3–21 (2008). https://doi.org/10.1016/j.scico.2007.08.001
9. Bakhirkin, A., Ferrère, T., Maler, O.: Efficient parametric identification for STL. In: HSCC, pp. 177–186. ACM (2018). https://doi.org/10.1145/3178126.3178132
10. Bakhirkin, A., Ferrère, T., Maler, O., Ulus, D.: On the quantitative semantics of regular expressions over real-valued signals. In: Abate, A., Geeraerts, G. (eds.) FORMATS 2017. LNCS, vol. 10419, pp. 189–206. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65765-3_11
11. Bakhirkin, A., Ferrère, T., Nickovic, D., Maler, O., Asarin, E.: Online timed pattern matching using automata. In: Jansen, D.N., Prabhakar, P. (eds.) FORMATS 2018. LNCS, vol. 11022, pp. 215–232. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00151-3_13
12. Barringer, H., Falcone, Y., Havelund, K., Reger, G., Rydeheard, D.: Quantified event automata: towards expressive and efficient runtime monitors. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 68–84. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32759-9_9
13. Basin, D.A., Klaedtke, F., Marinovic, S., Zalinescu, E.: Monitoring of temporal first-order properties with aggregations. Form. Methods Syst. Des. **46**(3), 262–285 (2015). https://doi.org/10.1007/s10703-015-0222-7
14. Basin, D.A., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring metric first-order temporal properties. J. ACM **62**(2), 15:1–15:45 (2015). https://doi.org/10.1145/2699444
15. Basin, D.A., Klaedtke, F., Zalinescu, E.: The MonPoly monitoring tool. In: Reger, G., Havelund, K. (eds.) RV-CuBES. Kalpa Publications in Computing, vol. 3, pp. 19–28. EasyChair (2017)
16. Bouajjani, A., Echahed, R., Robbana, R.: On the automatic verification of systems with continuous variables and unbounded discrete data structures. In: Antsaklis, P., Kohn, W., Nerode, A., Sastry, S. (eds.) HS 1994. LNCS, vol. 999, pp. 64–85. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60472-3_4
17. Bozzelli, L., La Torre, S.: Decision problems for lower/upper bound parametric timed automata. Form. Methods Syst. Des. **35**(2), 121–151 (2009). https://doi.org/10.1007/s10703-009-0074-0
18. Brim, L., Dluhos, P., Safránek, D., Vejpustek, T.: STL*: extending signal temporal logic with signal-value freezing operator. Inf. Comput. **236**, 52–67 (2014). https://doi.org/10.1016/j.ic.2014.01.012
19. Dang, Z.: Pushdown timed automata: a binary reachability characterization and safety verification. Theor. Comput. Sci. **302**(1–3), 93–121 (2003). https://doi.org/10.1016/S0304-3975(02)00743-0
20. Deshmukh, J.V., Majumdar, R., Prabhu, V.S.: Quantifying conformance using the Skorokhod metric. Form. Methods Syst. Des. **50**(2–3), 168–206 (2017). https://doi.org/10.1007/s10703-016-0261-8
21. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_17

22. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_19
23. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
24. Fages, F., Rizk, A.: On temporal logic constraint solving for analyzing numerical data time series. Theor. Comput. Sci. **408**(1), 55–65 (2008). https://doi.org/10.1016/j.tcs.2008.07.004
25. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theor. Comput. Sci. **410**(42), 4262–4291 (2009). https://doi.org/10.1016/j.tcs.2009.06.021
26. Havelund, K., Peled, D., Ulus, D.: First order temporal logic monitoring with BDDs. In: Stewart, D., Weissenbacher, G. (eds.) FMCAD, pp. 116–123. IEEE (2017). https://doi.org/10.23919/FMCAD.2017.8102249
27. Jakšić, S., Bartocci, E., Grosu, R., Nguyen, T., Ničković, D.: Quantitative monitoring of STL with edit distance. Form. Methods Syst. Des. **53**(1), 83–112 (2018). https://doi.org/10.1007/s10703-018-0319-x
28. Jovanović, A., Lime, D., Roux, O.H.: Integer parameter synthesis for real-time systems. IEEE Trans. Softw. Eng. **41**(5), 445–461 (2015). https://doi.org/10.1109/TSE.2014.2357445
29. Quaas, K.: Verification for timed automata extended with discrete data structure. Log. Methods Comput. Sci. **11**(3) (2015). https://doi.org/10.2168/LMCS-11(3:20)2015
30. Reger, G., Cruz, H.C., Rydeheard, D.: MarQ: monitoring at runtime with QEA. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 596–610. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_55
31. Ulus, D.: Montre: a tool for monitoring timed regular expressions. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017, Part I. LNCS, vol. 10426, pp. 329–335. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_16
32. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Timed pattern matching. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 222–236. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10512-3_16
33. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Online timed pattern matching using derivatives. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 736–751. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_47
34. Ulus, D., Maler, O.: Specifying timed patterns using temporal logic. In: HSCC, pp. 167–176. ACM (2018). https://doi.org/10.1145/3178126.3178129
35. Waga, M., André, É., Hasuo, I.: Symbolic monitoring against specifications parametric in time and data. CoRR abs/1905.04486 (2019). arxiv:1905.04486
36. Waga, M., Hasuo, I., Suenaga, K.: Efficient online timed pattern matching by automata-based skipping. In: Abate, A., Geeraerts, G. (eds.) FORMATS 2017. LNCS, vol. 10419, pp. 224–243. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65765-3_13

STAMINA: STochastic Approximate Model-Checker for INfinite-State Analysis

Thakur Neupane[1(✉)], Chris J. Myers[2], Curtis Madsen[3], Hao Zheng[4], and Zhen Zhang[1]

[1] Utah State University, Logan, UT, USA
thakur.neupane@aggiemail.usu.edu
zhen.zhang@usu.edu
[2] University of Utah, Salt Lake City, UT, USA
myers@ece.utah.edu
[3] Boston University, Boston, MA, USA
ckmadsen@bu.edu
[4] University of South Florida, Tampa, FL, USA
haozheng@usf.edu

Abstract. Stochastic model checking is a technique for analyzing systems that possess probabilistic characteristics. However, its scalability is limited as probabilistic models of real-world applications typically have very large or infinite state space. This paper presents a new infinite state CTMC model checker, STAMINA, with improved scalability. It uses a novel state space approximation method to reduce large and possibly infinite state CTMC models to finite state representations that are amenable to existing stochastic model checkers. It is integrated with a new property-guided state expansion approach that improves the analysis accuracy. Demonstration of the tool on several benchmark examples shows promising results in terms of analysis efficiency and accuracy compared with a state-of-the-art CTMC model checker that deploys a similar approximation method.

Keywords: Stochastic model checking · Infinite-state · Markov chains

1 Introduction

Stochastic model checking is a formal method that designers and engineers can use to determine the likelihood of *safety* and *liveness* properties. Checking properties using numerical model checking techniques requires enumerating the state space of the system to determine the probability that the system is in any given state at a desired time [17]. Real-world applications often have very large or even infinite state spaces.

Numerous state representation, reduction, and approximation methods have been proposed. Symbolic model checking based on *multi-terminal binary decision diagrams* (MTBDDs) [23] has achieved success in representing large *Markov Decision Process* (MDP) models with a few distinct probabilistic choices at each state, e.g., the shared coin protocol [3]. MTBDDs, however, are often inefficient for models with many different and distinct probability/rate values due to the inefficient representation of solution

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 540–549, 2019.
https://doi.org/10.1007/978-3-030-25540-4_31

vectors. *Continuous-time Markov chain* (CTMC) models, whose state transition rate is a function of state variables, generally contain many distinct rate values. As a result, symbolic model checkers can run out of memory while verifying a typical CTMC model with as few as 73,000 states [23]. State reduction techniques, such as bisimulation minimization [7,8,14], abstraction [6,12,14,20], symmetry reduction [5,16], and partial order reduction [9] have been mainly extended to discrete-time, finite-state probabilistic systems. The three-valued abstraction [14] can reduce large, finite-state CTMCs. It may, however, provide inconclusive verification results due to abstraction.

To the best of our knowledge, only a few tools can analyze infinite-state probabilistic models, namely, STAR [19] and INFAMY [10]. The STAR tool primarily analyzes biochemical reaction networks. It approximates solutions to the *chemical master equation* (CME) using the *method of conditional moments* (MCM) [11] that combines moment-based and state-based representations of probability distributions. This hybrid approach represents species with low concentrations using a discrete stochastic description and numerically integrates a small master equation using the fourth order Runge-Kutta method over a small time interval [2]; and solves a system of conditional moment equations for higher concentration species, conditioned on the low concentration species. This method has been optimized to drop unlikely states and add likely states on-the-fly. STAR relies on a well-structured underlying Markov process with small sensitivity on the transient distribution. Also, it mainly reports state reachability probabilities, instead of checking a given probabilistic property. INFAMY is a truncation-based approach that explores the model's state space up to a certain finite depth k. The truncated state space still grows exponentially with respect to exploration depth. Starting from the initial state, breadth-first state search is performed up to a certain finite depth. The error probability computed during the model checking depends on the depth of state exploration. Therefore, higher exploration depth generally incurs lower error probability.

This paper presents a new infinite-state stochastic model checker, *STochastic Approximate Model-checker for INfinite-state Analysis* (STAMINA). Our tool also takes a truncation-based approach. In particular, it maintains a probability estimate of each path being explored in the state space, and when the currently explored path probability drops below a specified threshold, it halts exploration of this path. All transitions exiting this state are redirected to an absorbing state. After all paths have been explored or truncated, transient Markov chain analysis is applied to determine the probability of a transient property of interest specified using *Continuous Stochastic Logic* (CSL) [4]. The calculated probability forms a lower bound on the probability, while the upper bound also includes the probability of the absorbing state. The actual probability of the CSL property is guaranteed to be within this range. An initial version of our tool and preliminary results are reported in [22]. Since that paper, our tool has been tightly integrated within the PRISM model checker [18] to improve performance, and we have also developed a new property-guided state expansion technique to expand the state space to tighten the reported probability range incrementally. This paper reports our results, which show significant improvement on both efficiency and verification accuracy over several non-trivial case studies from various application domains.

2 STAMINA

Figure 1 presents the architecture of STAMINA. Based on a user-specified probability threshold $\varkappa$ (kappa), it first constructs a finite-state CTMC model $\mathcal{C}\rfloor_{\varkappa}$ from the original infinite-state CTMC model $\mathcal{C}$ using the state space approximation method presented in Sect. 2.1. $\mathcal{C}\rfloor_{\varkappa}$ is then checked using the PRISM explicit-state model checker against a given CSL property $\mathsf{P}_{\sim p}(\phi)$, where $\sim \in \{<, >, \leqslant, \geqslant\}$ and $p \in [0, 1]$ (for cases where it is desired that a predicate be true within a certain probability bound) or $\mathsf{P}_{=?}(\phi)$ (for cases where it is desired that the exact probability of the predicate being true be calculated). Lower- and upper-bound probabilities that ϕ holds, namely, P_{min} and P_{max}, are then obtained, and their difference, i.e., $(P_{max} - P_{min})$, is the probability accumulated in the absorbing state $\mathbf{x}_{abs}$ which abstracts all the states not included in the current state space. If $p \in [P_{min}, P_{max}]$, it is not known whether $\mathsf{P}_{\sim p}(\phi)$ holds. If exact probability is of interest and the probability range is larger than the user-defined precision ϵ, i.e., $(P_{max} - P_{min}) > \epsilon$, then the method does not give a meaningful result.

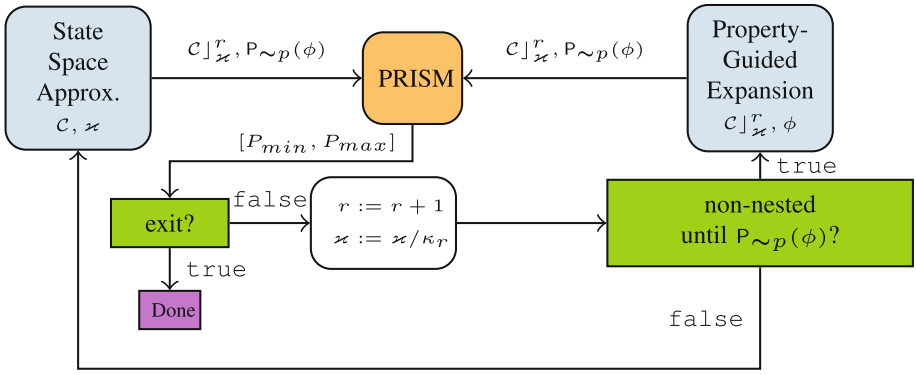


Fig. 1. Architecture of STAMINA.

For an inconclusive verification result from the previous step, STAMINA applies a property-guided approach, described in Sect. 2.2, to further expand $\mathcal{C}\rfloor_{\varkappa}$, provided $\mathsf{P}_{\sim p}(\phi)$ is a non-nested "until" formula; otherwise, it uses the previous method to expand the state space. Note that $\varkappa$ also drops by the reduction factor κ_r to enable states that were previously ignored due to a low probability estimate to be included in the current state expansion. The expanded CTMC model $\mathcal{C}\rfloor_{\varkappa}$ is then checked to obtain a new probability bound $[P_{min}, P_{max}]$. This iterative process repeats until one of the following conditions holds: (1) the target probability p falls outside the probability bound $[P_{min}, P_{max}]$, (2) the probability bound is sufficiently small, i.e, $(P_{max} - P_{min}) < \epsilon$, or (3) a maximal number of iterations N has been reached ($r \geqslant N$).

2.1 State Space Approximation

The state space approximation method [22] truncates the state space based on a user-specified reachability threshold $\varkappa$. During state exploration, the reachability-value func-

tion, $\hat{\kappa} : \mathbf{X} \to \mathbb{R}^+$, estimates the probability of reaching a state on-the-fly, and is compared against $\varkappa$ to determine whether the state search should terminate. Only states with a higher reachability-value than the reachability threshold are explored further.

Figure 2 illustrates the standard *breadth first search* (BFS) state exploration for reachability threshold $\varkappa = 0.25$. It starts from the initial state whose reachability-value i.e., $\hat{\kappa}(\mathbf{x}_0)$, is initialized to 1.0 as shown in Fig. 2a. In the first step, two new states $\mathbf{x}_1$ and $\mathbf{x}_4$ are generated and their reachability-values are 0.8 and 0.2, respectively, as shown in Fig. 2b. The reachability-value in $\mathbf{x}_0$ is distributed to its successor states, based on the probability of outgoing transitions from $\mathbf{x}_0$ to its successor state. For the next step, only state $\mathbf{x}_1$ is scheduled for exploration because $\hat{\kappa}(\mathbf{x}_1) \geq \varkappa$. Note that the transition from $\mathbf{x}_4$ to $\mathbf{x}_0$ is executed because $\mathbf{x}_0$ is already in the explored set. Expanding $\mathbf{x}_1$ leads to two new states, namely $\mathbf{x}_2$ and $\mathbf{x}_5$ as shown in Fig. 2c, from which only $\mathbf{x}_5$ is scheduled for further exploration. This leads to the generation of $\mathbf{x}_6$ and $\mathbf{x}_9$ shown in Fig. 2d. State exploration terminates after Fig. 2e since both newly generated states have reachability-values less than 0.25. States $\mathbf{x}_2$, $\mathbf{x}_4$, $\mathbf{x}_6$ and $\mathbf{x}_9$ are marked as terminal states. During state exploration, the reachability-value update is performed every time a new incoming path is added to a state because a new incoming path can add its contribution to the state, potentially bringing the reachability-value above $\varkappa$, which in turn changes a terminal state to be non-terminal. When the truncated CTMC model $\mathcal{C}\rfloor_{\varkappa}$ is analyzed, it introduces some error in the probability value of the property under verification, because of leakage the probability (i.e., cumulative path probabilities of reaching states not included in the explored state space) during the CTMC analysis. To

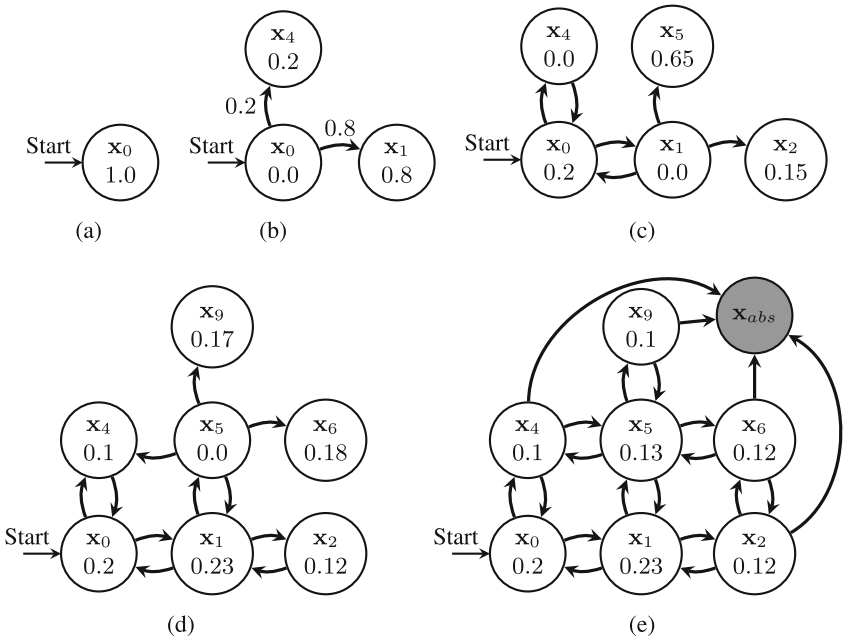


Fig. 2. State space approximation.

account for probability loss, an abstract absorbing state $\mathbf{x}_{abs}$ is created as the sole successor state for all terminal states on each truncated path. Figure 2e shows the addition of the absorbing state.

2.2 Property Based State Space Exploration

This paper introduces a property-guided state expansion method, in order to efficiently obtain a tightened probability bound. Since all non-nested CSL path formulas ϕ (except those containing the "next" operator) derive from the "until" formula, $\Phi\ \mathcal{U}^I\ \Psi$, construction of the set of terminal states for further expansion boils down to eliminating states that are known to satisfy or dissatisfy $\Phi\ \mathcal{U}\ \Psi$. Given a state graph, a path starting from the initial state can never satisfy $\Phi\ \mathcal{U}\ \Psi$, if it includes a state satisfying $\neg\Phi \wedge \neg\Psi$. Also, if a path includes a state satisfying Ψ, satisfiability of $\Phi\ \mathcal{U}\ \Psi$ can be determined without further expanding this path beyond the first Ψ-state. Our property-guided state space expansion method identifies the path prefixes, from which satisfiability of $\Phi\ \mathcal{U}\ \Psi$ can be determined, and shortens them by making the last state of each prefix absorbing based on the satisfiability of $(\neg\Phi \vee \Psi)$. Only the non-absorbing states whose path probability is greater than the state probability estimate threshold $\varkappa$ are expanded further. For detailed algorithms of STAMINA, readers are encouraged to read [21].

3 Results

This section presents results on the following case studies to illustrate the accuracy and efficiency of STAMINA: a genetic toggle switch [20,22]; the following examples from the PRISM benchmark suite [15]: grid world robot, cyclic server polling system, and tandem queuing network; and the Jackson queuing network from INFAMY case studies [1]. All case studies are evaluated on STAMINA and INFAMY, except the genetic toggle switch [1]. Experiments are performed on a 3.2 GHz AMD Debian Linux PC with six cores and 64 GB of RAM. For all experiments, the maximal number of iterations N is set to 10, and the reduction factor κ_r is set to 1000. All experiments terminate due to $(P_{max} - P_{min}) < \epsilon$, where $\epsilon = 10^{-3}$, before they reach N. STAMINA is freely available at: https://github.com/formal-verification-research/stamina.

We compare the runtime, state size, and verification results between STAMINA and INFAMY using the same precision $\epsilon = 10^{-3}$. For all tables in this section, column $\varkappa$ reports the probability estimate threshold used to terminate state generation in STAMINA. The state space size is listed in column $|\mathcal{G}|(K)$, where K indicates one thousand states. Column $T(C/A)$ reports the state space construction (C) and analysis (A) time in seconds. For STAMINA, the total construction and analysis time is the cumulation of runtime for all $\varkappa$ values for a model configuration. Columns P_{min} and P_{max} list the lower and upper probability bounds for the property under verification, and column P lists the single probability value (within the precision ϵ) reported by INFAMY. We select the best runtime reported by three configurations of INFAMY. The improvement in state size (column $|\mathcal{G}|(X)$) and runtime (column $T(\%)$) are represented

[1] INFAMY generates arithmetic errors on the genetic toggle switch model.

by the ratio of state count generated by INFAMY to that of STAMINA (higher is better) and percentage improvement in runtime (higher is better), respectively.

Genetic Toggle Switch. The genetic toggle switch circuit model has two inputs, aTc and IPTG. It can be set to the OFF state by supplying it with aTc and can be set to the ON state by supplying it with IPTG [20]. Two important properties for a toggle switch circuit are the response time and the failure rate. The first experiments set IPTG to 100 to measure the toggle switch's response time. It should be noted that the input value of 100 molecules of IPTG is chosen to ensure that the circuit switches to the ON state. The later experiments initialize IPTG to 0 to compute the failure rate, i.e., the probability that the circuit changes state erroneously within a cell cycle of $2,100$ s (an approximation of the cell cycle in *E. coli* [24]). Initially, LacI is set to 60 and TetR is set to 0 for both experiments. The CSL property used for both experiments, $\mathsf{P}_{=?}\ [true\ \mathcal{U}^{\leqslant 2100}\ (TetR > 40\ \wedge\ LacI < 20)]$, describes the probability of the circuit switching to the ON state within a cell cycle of $2,100$ s. The ON state is defined as LacI below 20 and TetR above 40 molecules.

Table 1. Verification results for genetic toggle switch.

$IPTG$	STAMINA					
	$\varkappa$	$\lvert\mathcal{G}\rvert$	$T(C/A)$	P_{min}	P_{max}	Remark
100	10^{-3}	1,127	0.15/0.67	0.000000	0.999671	Property guided
	10^{-6}	4,461	0.43/2.84	0.966947	0.992908	
	10^{-9}	7,163	0.43/5.25	0.991738	0.991797	
100	10^{-6}	5,171	0.17/1.90	0.977942	0.992850	Property agnostic
	10^{-9}	8,908	0.18/3.74	0.991739	0.991797	
0	10^{-3}	182	0.05/0.07	0.000000	0.697500	Property guided
	10^{-6}	2,438	0.16/1.08	0.008814	0.060424	
	10^{-9}	4,284	0.09/2.12	0.013097	0.013609	
0	10^{-6}	2,446	0.16/1.05	0.009169	0.060420	Property agnostic
	10^{-9}	4,820	0.13/2.13	0.013097	0.013609	

The property-agnostic state space is generated with the probability estimate threshold $\varkappa = 10^{-3}$. Table 1 shows large probability bounds: $[0, 0.999671]$ for IPTG $= 100$ and $[0, 0.6975]$ for IPTG $= 0$. It is obvious that they are significantly inaccurate w.r.t. the precision ϵ of 10^{-3}. The $\varkappa$ is then reduced to 10^{-6} and state generation switches to the property-guided state expansion mode, where the CSL property is used to guide state exploration, based on the previous state graph. Each state expansion step reduces the $\varkappa$ value by a factor of $\kappa_r = 1000$. To measure the effectiveness of the property-guided state expansion approach, we compare state graphs generated with and without the property-guided state expansion, as indicated by the "property agnostic" and "property guided" rows in the table. Property-guided state expansion reduces the size of the state space without losing the analysis precision for the same value of $\varkappa$. Specifically,

the state expansion approach reduces the state space by almost 20% for the response rate experiment.

Robot World. This case study considers a robot moving in an n-by-n grid and a janitor moving in a larger grid Kn-by-Kn, where the constant K is used to significantly scale up the state space. The robot starts from the bottom left corner to reach the top right corner. The janitor moves around randomly. Either the robot or janitor can occupy one grid location at any given time. The robot also randomly communicates with the base station. The property of interest is the probability that the robot reaches the top right corner within 100 time units while periodically communicating with the base station, encoded as $\mathsf{P}_{=?}\,[\,(\mathsf{P}_{\geqslant 0.5}\,[\,true\;\mathcal{U}^{\leqslant 7}\;communicate\,])\;\mathcal{U}^{\leqslant 100}\;goal\,]$.

Table 2 provides a comparison of results for $K = 1024, 64$ and $n = 64, 32$. For smaller grid size i.e, 32-by-32, the robot can reach the goal with a high probability of 97.56%. Where as for a larger value of $n = 64$ and $K = 64$, the robot is not able to reach the goal with considerable probability. STAMINA generates precise results that are similar to INFAMY, while exploring less than half of states with shorter runtime.

Table 2. Comparison between STAMINA and INFAMY.

Model	Params	STAMINA				INFAMY			Improvement	
		$\lvert\mathcal{G}\rvert\,(K)$	$T\,(C/A)$	P_{min}	P_{max}	$\lvert\mathcal{G}\rvert\,(K)$	$T\,(C/A)$	P	$\lvert\mathcal{G}\rvert\,(X)$	$T\,(\%)$
Robot (n/K)	32/64	696	41/279	0.975	0.975	1,591	492/18	0.975	2.3	37.3
	32/1024	696	41/258	0.975	0.975	1,591	501/18	0.975	2.3	42.4
	64/64	2,273	135/669	$1.46e{-4}$	$1.68e{-4}$	5,088	1,625/53	$1.5e{-4}$	2.2	52.1
	64/1024	2,273	132/621	$1.46e{-4}$	$1.68e{-4}$	5,088	1,625/53	$1.5e{-4}$	2.2	55.2
Jackson (N/λ)	4/5	201	22/51	0.865	0.865	635	109/5	0.865	3.2	36.1
	5/5	2,539	990/996	0.819	0.819	7,029	1668/108	0.819	2.8	−11.8
Polling (N)	12	19	3/21	1.0	1.0	74	1/2	1.0	3.9	−732.2
	16	57	18/70	1.0	1.0	1,573	5/54	1.0	27.6	−48.2
	20	113	30/77	1.0	1.0	31,457	151/1347	1.0	278.4	92.9
Tandem (c)	2047	33	1/41	0.498	0.498	2,392	3/38	0.498	72.5	−1.4
	4095	66	1/141	0.499	0.499	9,216	11/265	0.499	139.6	48.7

Jackson Queuing Network. A Jackson queuing network consists of N interconnected nodes (queues) with infinite queue capacity. Initially, all queues are considered empty. Each station is connected to a single server which distributes the arrived jobs to different stations. Customers arrive as a Poisson stream with intensity λ for N queues. The model is taken from [10,13]. We compute the probability that, within 10 time units, the first queue has more that 3 jobs and the second queue has more than 5 jobs, given by $\mathsf{P}_{=?}\,[\,true\;\mathcal{U}^{\leqslant 10}\;(jobs\_1 \geqslant 4\;\wedge\;jobs\_2 \geqslant 6)]$.

Table 2 summarizes the results for this model. STAMINA uses roughly equal time to construct and analyze the model for $N = 5$, whereas INFAMY takes significantly longer to construct the state space, making it slower in overall runtime. For $N = 4$, STAMINA is faster in generating verification results In both configurations, STAMINA only explores approximately one third of the states explored by INFAMY.

Cyclic Server Polling System. This case study is based on a cyclic server attending N stations. We consider the probability that station one is polled within 10 time units, $\mathsf{P}_{=?}\ [\ true\ \mathcal{U}^{\leqslant 10}\ station1\_polled\ ]$. Table 2 summarizes the verification results for $N = 12, 16, 20$. The probability of station one being polled within 10 s is 1.0 for all configurations. Similar to previous case studies, STAMINA explores significantly smaller state space. The advantage of STAMINA in terms of runtime starts to manifest as the size of model (and hence the state space size) grows.

Tandem Queuing Network. A tandem queuing network is the simplest interconnected queuing network of two finite capacity (c) queues with one server each [18]. Customers join the first queue and enter the second queue immediately after completing the service. This paper considers the probability that the first queue becomes full in 0.25 time units, depicted by the CSL property $\mathsf{P}_{=?}\ [\ true\ \mathcal{U}^{\leqslant 0.25}\ queue1\_full\ ]$.

As seen in Table 2, there is almost fifty percent probability that the first queue is full in 0.25 s irrespective of the queue capacity. As in the polling server, STAMINA explores significantly smaller state space. The runtime is similar for model with smaller queue capacity ($c = 2047$). But the runtime improves as the queue capacity is increased.

4 Conclusions

This paper presents an infinite-state stochastic model checker, STAMINA, that uses path probability estimates to generate states with high probability and truncate unlikely states based on a specified threshold. Initial state construction is property agnostic, and the state space is used for stochastic model checking of a given CSL property. The calculated probability forms a lower and upper bound on the probability for the CSL property, which is guaranteed to include the actual probability. Next, if finer precision of the probability bound is required, it uses a property-guided state expansion technique to explore states to tighten the reported probability range incrementally. Implementation of STAMINA is built on top of the PRISM model checker with tight integration to its API. Performance and accuracy evaluation is performed on case studies taken from various application domains, and shows significant improvement over the state-of-art infinite-state stochastic model checker INFAMY. For future work, we plan to investigate methods to determine the reduction factor on-the-fly based on the probability bound. Another direction is to investigate heuristics to further improve the property-guided state expansion, as well as, techniques to dynamically remove unlikely states.

Acknowledgment. Chris Myers is supported by the National Science Foundation under CCF-1748200. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

1. https://depend.cs.uni-saarland.de/tools/infamy/casestudies/
2. Andreychenko, A., Mikeev, L., Spieler, D., Wolf, V.: Parameter identification for Markov models of biochemical reactions. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 83–98. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_8

3. Aspnes, J., Herlihy, M.: Fast randomized consensus using shared memory. J. Algorithms **11**(3), 441–461 (1990)
4. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time Markov chains. ACM Trans. Comput. Logic **1**(1), 162–170 (2000)
5. Donaldson, A.F., Miller, A.: Symmetry reduction for probabilistic model checking using generic representatives. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 9–23. Springer, Heidelberg (2006). https://doi.org/10.1007/11901914_4
6. Fecher, H., Leucker, M., Wolf, V.: *Don't know* in probabilistic systems. In: Valmari, A. (ed.) SPIN 2006. LNCS, vol. 3925, pp. 71–88. Springer, Heidelberg (2006). https://doi.org/10.1007/11691617_5
7. Fisler, K., Vardi, M.Y.: Bisimulation and model checking. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 338–342. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48153-2_29
8. Fisler, K., Vardi, M.Y.: Bisimulation minimization and symbolic model checking. Form. Methods Syst. Des. **21**(1), 39–78 (2002)
9. Groesser, M., Baier, C.: Partial order reduction for Markov decision processes: a survey. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2005. LNCS, vol. 4111, pp. 408–427. Springer, Heidelberg (2006). https://doi.org/10.1007/11804192_19
10. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: INFAMY: an infinite-state Markov model checker. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 641–647. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_49
11. Hasenauer, J., Wolf, V., Kazeroonian, A., Theis, F.J.: Method of conditional moments (MCM) for the chemical master equation. J. Math. Biol. **69**(3), 687–735 (2014). https://doi.org/10.1007/s00285-013-0711-5
12. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 162–175. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70545-1_16
13. Jackson, J.: Networks of waiting lines. Oper. Res. **5**, 518–521 (1957)
14. Katoen, J.-P., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for continuous-time Markov chains. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 311–324. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_37
15. Kwiatkowsa, M., Norman, G., Parker, D.: The prism benchmark suite. In: International Conference on (QEST) Quantitative Evaluation of Systems, vol. 00, pp. 203–204, September 2012. https://doi.org/10.1109/QEST.2012.14, doi.ieeecomputersociety.org/10.1109/QEST.2012.14
16. Kwiatkowska, M., Norman, G., Parker, D.: Symmetry reduction for probabilistic model checking. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 234–248. Springer, Heidelberg (2006). https://doi.org/10.1007/11817963_23
17. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72522-0_6
18. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
19. Lapin, M., Mikeev, L., Wolf, V.: Shave: Stochastic hybrid analysis of Markov population models. In: Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control, HSCC 2011, pp. 311–312. ACM, New York (2011)
20. Madsen, C., Zhang, Z., Roehner, N., Winstead, C., Myers, C.: Stochastic model checking of genetic circuits. J. Emerg. Technol. Comput. Syst. **11**(3), 23:1–23:21 (2014). https://doi.org/10.1145/2644817. http://doi.acm.org/10.1145/2644817

21. Neupane, T.: STAMINA: STochastic approximate model-checker for INfinite-state analysis. Master's thesis, Utah State University, May 2019
22. Neupane, T., Zhang, Z., Madsen, C., Zheng, H., Myers, C.J.: Approximation techniques for stochastic analysis of biological systems. In: Liò, P., Zuliani, P. (eds.) Automated Reasoning for Systems Biology and Medicine. CB, vol. 30, pp. 327–348. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17297-8_12
23. Parker, D.: Implementation of symbolic model checking for probabilistic systems. Ph.D. thesis, University of Birmingham (2002)
24. Zheng, H., et al.: Interrogating the escherichia coli cell cycle by cell dimension perturbations. Proc. Natl. Acad. Sci. **113**(52), 15000–15005 (2016)

Dynamical, Hybrid, and Reactive Systems

Local and Compositional Reasoning for Optimized Reactive Systems

Mitesh Jain[1(✉)] and Panagiotis Manolios[2]

[1] Synopsys Inc., Mountain View, USA
mitesh.jain@synopsys.com
[2] Northeastern University, Boston, USA
pete@ccs.neu.edu

Abstract. We develop a compositional, algebraic theory of skipping refinement, as well as local proof methods to effectively analyze the correctness of optimized reactive systems. A verification methodology based on refinement involves showing that any infinite behavior of an optimized low-level implementation is a behavior of the high-level abstract specification. Skipping refinement is a recently introduced notion to reason about the correctness of optimized implementations that run faster than their specifications, *i.e.*, a step in the implementation can skip multiple steps of the specification. For the class of systems that exhibit bounded skipping, existing proof methods have been shown to be amenable to mechanized verification using theorem provers and model-checkers. However, reasoning about the correctness of reactive systems that exhibit unbounded skipping using these proof methods requires reachability analysis, significantly increasing the verification effort. In this paper, we develop two new sound and complete proof methods for skipping refinement. Even in presence of unbounded skipping, these proof methods require only local reasoning and, therefore, are amenable to mechanized verification. We also show that skipping refinement is compositional, so it can be used in a stepwise refinement methodology. Finally, we illustrate the utility of the theory of skipping refinement by proving the correctness of an optimized event processing system.

1 Introduction

Reasoning about the correctness of a reactive system using refinement involves showing that any (infinite) observable behavior of a low-level, optimized implementation is a behavior allowed by the simple, high-level abstract specification. Several notions of refinement like trace containment, (bi)simulation refinement, stuttering (bi)simulation refinement, and skipping refinement [4,10,14,20,22] have been proposed in the literature to directly account for the difference in the abstraction levels between a specification and an implementation. Two attributes of crucial importance that enable us to effectively verify complex reactive systems using refinement are: (1) Compositionality: this allows us to decompose a monolithic proof establishing that a low-level concrete implementation refines

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 553–571, 2019.
https://doi.org/10.1007/978-3-030-25540-4_32

a high-level abstract specification into a sequence of simpler refinement proofs, where each of the intermediate refinement proof can be performed independently using verification tools best suited for it; (2) Effective proof methods: analyzing the correctness of a reactive system requires global reasoning about its infinite behaviors, a task that is often difficult for verification tools. Hence it is crucial that the refinement-based methodology also admits effective proof methods that are amenable for mechanized reasoning.

It is known that the (bi)simulation refinement and stuttering (bi)simulation refinement are compositional and support the stepwise refinement methodology [20,24]. Moreover, the proof methods associated with them are local, *i.e.*, they only require reasoning about states and their successors. Hence, they are amenable to mechanized reasoning. However, to the best of our knowledge, it is not known if skipping refinement is compositional. Skipping refinement is a recently introduced notion of refinement for verifying the correctness of optimized implementations that can "execute faster" than their simple high-level specifications, *i.e.*, a step in the implementation can *skip* multiple steps in the specification. Examples of such systems include superscalar processors, concurrent and parallel systems and optimizing compilers. Two proof methods, *reduced well-founded skipping simulation* and *well-founded skipping simulation* have been introduced to reason about skipping refinement for the class of systems that exhibit bounded skipping [10]. These proof methods were used to verify the correctness of several systems that otherwise were difficult to automatically verify using current model-checkers and automated theorem provers. However, when skipping is unbounded, the proof methods in [10] require reachability analysis, and therefore are not amenable to automated reasoning. To motivate the need for alternative proof methods for effective reasoning, we consider the event processing system (EPS), discussed in [10].

1.1 Motivating Example

An abstract high-level specification, AEPS, of an event processing system is defined as follows. Let E be a set of *events* and V be a set of *state variables*. A *state* of AEPS is a triple $\langle t, Sch, St\rangle$, where t is a natural number denoting the current time; Sch is a set of pairs $\langle e, t_e\rangle$, where $e \in E$ is an event scheduled to be executed at time $t_e \geq t$; St is an assignment to state variables in V. The transition relation for the AEPS system is defined as follows. If at time t there is no $\langle e, t\rangle \in Sch$, *i.e.*, there is no event scheduled to be executed at time t, then t is incremented by 1. Otherwise, we (nondeterministically) choose and execute an event of the form $\langle e, t\rangle \in Sch$. The execution of an event may result in modifying St and also removing and adding a finite number of new pairs $\langle e', t'\rangle$ to Sch. We require that $t' > t$. Finally, execution involves removing the executed event $\langle e, t\rangle$ from Sch. Now consider, tEPS, an optimized implementation of AEPS. As before, a state is a triple $\langle t, Sch, St\rangle$. However, unlike the abstract system which just increments time by 1 when there are no events scheduled at the current time, the optimized system finds the earliest time in future an event is scheduled to execute. The transition relation of tEPS is defined as follows. An event (e, t_e)

with the minimum time is selected, t is updated to t_e and the event e is executed, as in the AEPS. Consider an execution of AEPS and tEPS in Fig. 1. (We only show the prefix of executions). Suppose at $t = 0$, Sch be $\{(e_1, 0)\}$. The execution of event e_1 add a new pair (e_2, k) to Sch, where k is a positive integer. AEPS at $t = 0$, executes the event e_1, adds a new pair (e_2, k) to Sch, and updates t to 1. Since no events are scheduled to execute before $t = k$, the AEPS system repeatedly increments t by 1 until $t = k$. At $t = k$, it executes the event e_2. At time $t = 0$, tEPS executes e_1. The next event is scheduled to execute at time $t = k$; hence it updates in one step t to k. Next, in one step it executes the event e_2. Note that tEPS runs faster than AEPS by *skipping* over abstract states when no event is scheduled for execution at the current time. If $k > 1$, the step from s_2 to s_3 in tEPS neither corresponds to stuttering nor to a single step of the AEPS. Therefore notions of refinement based on stuttering simulation and bisimulation cannot be used to show that tEPS refines AEPS.

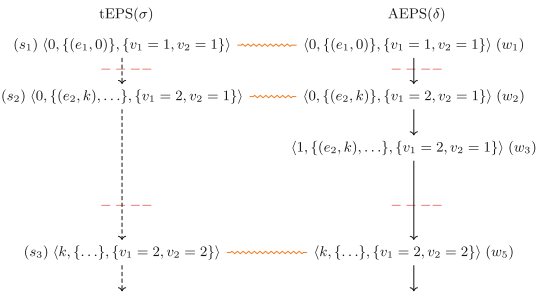


Fig. 1. Event simulation system

It was argued in [10] that skipping refinement is an appropriate notion of correctness that directly accounts for the skipping behavior exhibited by tEPS. Though, tEPS was used to motivate the need for a new notion of refinement, the proof methods proposed in [10] are not effective to prove the correctness of tEPS. This is because, execution of an event in tEPS may add new events that are scheduled to execute at an arbitrary time in future, *i.e.*, in general k in the above example execution is unbounded. Hence, the proof methods in [10] would require unbounded reachability analysis which often is problematic for automated verification tools. Even in the particular case when one can a priori determine an upper bound on k and unroll the transition relation, the proof methods in [10] are viable for mechanical reasoning only if the upper bound k is relatively small.

In this paper, we develop local proof methods to effectively analyze the correctness of optimized reactive systems using skipping refinement. These proof methods reduce global reasoning about infinite computations to local reasoning about states and their successor and are applicable even if the optimized implementation exhibits unbounded skipping. Moreover, we show that the proposed

proof methods are complete, *i.e.*, if a system $\mathcal{M}_1$ is a skipping refinement of $\mathcal{M}_2$ under a suitable refinement map, then we can always locally reason about them. We also develop an algebraic theory of skipping refinement. In particular, we show that skipping simulation is closed under relational composition. Thus, skipping refinement aligns with the stepwise refinement methodology. Finally, we illustrate the benefits of the theory of skipping refinement and the associated proof methods by verifying the correctness of optimized event processing systems in ACL2s [3].

2 Preliminaries

A transition system model of a reactive system captures the concept of a state, atomic transitions that modify state during the course of a computation, and what is observable in a state. Any system with a well defined operational semantics can be mapped to a labeled transition system.

Definition 1 Labeled Transition System. *A labeled transition system (TS) is a structure $\langle S, \rightarrow, L\rangle$, where S is a non-empty (possibly infinite) set of states, $\rightarrow \subseteq S \times S$, is a left-total transition relation (every state has a successor), and L is a labeling function whose domain is S.*

Notation: We first describe the notational conventions used in the paper. Function application is sometimes denoted by an infix dot "." and is left-associative. The composition of relation R with itself i times (for $0 < i \leq \omega$) is denoted R^i ($\omega = \mathbb{N}$ and is the first infinite ordinal). Given a relation R and $1 < k \leq \omega$, $R^{<k}$ denotes $\bigcup_{1 \leq i < k} R^i$ and $R^{\geq k}$ denotes $\bigcup_{\omega > i \geq k} R^i$. Instead of $R^{<\omega}$ we often write the more common R^+. $\uplus$ denotes the disjoint union operator. Quantified expressions are written as $\langle Qx\colon r\colon t\rangle$, where Q is the quantifier (*e.g.*, $\exists, \forall, min, \bigcup$), x is a bound variable, r is an expression that denotes the range of variable x (*true*, if omitted), and t is a term.

Let $\mathcal{M} = \langle S, \rightarrow, L\rangle$ be a transition system. An $\mathcal{M}$-path is a sequence of states such that for adjacent states, s and u, $s \rightarrow u$. The j^{th} state in an $\mathcal{M}$-path σ is denoted by $\sigma.j$. An $\mathcal{M}$-path σ starting at state s is a *fullpath*, denoted by $fp.\sigma.s$, if it is infinite. An $\mathcal{M}$-segment, $\langle v_1, \ldots, v_k\rangle$, where $k \geq 1$ is a finite $\mathcal{M}$-path and is also denoted by $\vec{v}$. The length of an $\mathcal{M}$-segment $\vec{v}$ is denoted by $|\vec{v}|$. Let *INC* be the set of strictly increasing sequences of natural numbers starting at 0. The i^{th} partition of a fullpath σ with respect to $\pi \in INC$, denoted by ${}^{\pi}\sigma^i$, is given by an $\mathcal{M}$-segment $\langle \sigma(\pi.i), \ldots, \sigma(\pi(i+1)-1)\rangle$.

3 Theory of Skipping Refinement

In this section we first briefly recall the notion of skipping simulation as described in [10]. We then study the algebraic properties of skipping simulation and show that a theory of refinement based on it is compositional and therefore can be used in a stepwise refinement based verification methodology.

The definition of skipping simulation is based on the notion of *matching*. Informally, a fullpath σ matches a fullpath δ under the relation B iff the fullpaths can be partitioned in to non-empty, finite segments such that all elements in a segment of σ are related to the first element in the corresponding segment of δ.

Definition 2 smatch [10]. *Let $\mathcal{M} = \langle S, \rightarrow, L\rangle$ be a transition system, σ, δ be fullpaths in $\mathcal{M}$. For $\pi, \xi \in INC$ and binary relation $B \subseteq S \times S$, we define*

$$scorr(B, \sigma, \pi, \delta, \xi) \equiv \langle \forall i \in \omega :: \langle \forall s \in {}^{\pi}\sigma^{i} :: sB\delta(\xi.i)\rangle\rangle \text{ and}$$
$$smatch(B, \sigma, \delta) \equiv \langle \exists \pi, \xi \in INC :: scorr(B, \sigma, \pi, \delta, \xi)\rangle.$$

Figure 1 illustrates the notion of matching using our running example: σ is the fullpath of the concrete system and δ is a fullpath of the absract system. (The figure only shows the prefix of the fullpaths). The other parameter for matching is the relation B, which is just the identity function. In order to show that $smatch(B, \sigma, \delta)$ holds, we have to find $\pi, \xi \in INC$ satisfying the definition. In Fig. 1, we separate the partitions induced by our choice for π, ξ using $--$ and connect elements related by B with $\frown$. Since all elements of a σ partition are related to the first element of the corresponding δ partition, $scorr(B, \sigma, \pi, \delta, \xi)$ holds, therefore, $smatch(B, \sigma, \delta)$ holds.

Using the notion of matching, skipping simulation is defined as follows. Notice that skipping simulation is defined using a single transition system; it is easy to lift the notion defined on a single transition system to one that relates two transition systems by taking the disjoint union of the transition systems.

Definition 3 Skipping Simulation (SKS). *$B \subseteq S \times S$ is a skipping simulation on a TS $\mathcal{M} = \langle S, \rightarrow, L\rangle$ iff for all s, w such that sBw, both of the following hold.*

(SKS1) $L.s = L.w$
(SKS2) $\langle \forall \sigma : fp.\sigma.s : \langle \exists \delta : fp.\delta.w : smatch(B, \sigma, \delta)\rangle\rangle$

Theorem 1. *Let $\mathcal{M}$ be a TS. If B is a stuttering simulation (STS) on $\mathcal{M}$ then B is an SKS on $\mathcal{M}$.*

Proof: Follows directly from the definitions of SKS and STS [18]. □

3.1 Algebraic Properties

We now study the algebraic properties of SKS. We show that it is closed under arbitrary union. We also show that SKS is closed under relational composition. The later property is particularly useful since it enables us to use stepwise refinement and to modularly analyze the correctness of complex systems.

Lemma 1. *Let $\mathcal{M}$ be a TS and $\mathcal{C}$ be a set of SKS's on $\mathcal{M}$. Then $G = \langle \cup B : B \in \mathcal{C} : B\rangle$ is an SKS on $\mathcal{M}$.*

Corollary 1. *For any TS $\mathcal{M}$, there is a greatest SKS on $\mathcal{M}$.*

Lemma 2. *SKS are not closed under negation and intersection.*

The following lemma shows that skipping simulation is closed under relational composition.

Lemma 3. *Let $\mathcal{M}$ be a TS. If P and Q are SKS's on $\mathcal{M}$, then $R = P;Q$ is an SKS on $\mathcal{M}$.*

Proof: To show that R is an SKS on $\mathcal{M} = \langle S, \rightarrow, L\rangle$, we show that for any $s, w \in S$ such that sRw, SKS1 and SKS2 hold. Let $s, w \in S$ and sRw. From the definition of R, there exists $x \in S$ such that sPx and xQw. Since P and Q are SKS's on $\mathcal{M}$, $L.s = L.x = L.w$, hence, SKS1 holds for R.

To prove that SKS2 holds for R, consider a fullpath σ starting at s. Since P and Q are SKSs on $\mathcal{M}$, there is a fullpath τ in $\mathcal{M}$ starting at x, a fullpath δ in $\mathcal{M}$ starting at w and $\alpha, \beta, \theta, \gamma \in \mathit{INC}$ such that $\mathit{scorr}(P, \sigma, \alpha, \tau, \beta)$ and $\mathit{scorr}(Q, \tau, \theta, \delta, \gamma)$ hold. We use the fullpath δ as a witness and define $\pi, \xi \in \mathit{INC}$ such that $\mathit{scorr}(R, \sigma, \pi, \delta, \xi)$ holds.

We define a function, r, that given i, corresponding to the index of a partition of τ under β, returns the index of the partition of τ under θ in which the first element of τ's i^{th} partition under β resides. $r.i = j$ iff $\theta.j \leq \beta.i < \theta(j+1)$. Note that r is indeed a function, as every element of τ resides in exactly one partition of θ. Also, since there is a correspondence between the partitions of α and β, (by $\mathit{scorr}(P, \sigma, \alpha, \tau, \beta)$), we can apply r to indices of partitions of σ under α to find where the first element of the corresponding β partition resides. Note that r is non-decreasing: $a < b \Rightarrow r.a \leq r.b$.

We define $\pi\alpha \in \mathit{INC}$, a strictly increasing sequence that will allow us to merge adjacent partitions in α as needed to define the strictly increasing sequence π on σ used to prove SKS2. Partitions in π will consist of one or more α partitions. Given i, corresponding to the index of a partition of σ under π, the function $\pi\alpha$ returns the index of the corresponding partition of σ under α.

$$\pi\alpha(0) = 0$$

$$\pi\alpha(i) = \text{ min } j \in \omega \text{ s.t. } |\{k : 0 < k \leq j \wedge r.k \neq r(k-1)\}| = i$$

Note that $\pi\alpha$ is an increasing function, *i.e.*, $a < b \Rightarrow \pi\alpha(a) < \pi\alpha(b)$. We now define π as follows.

$$\pi.i = \alpha(\pi\alpha.i)$$

There is an important relationship between r and $\pi\alpha$

$$r(\pi\alpha.i) = \cdots = r(\pi\alpha(i+1) - 1)$$

That is, for all α partitions that are in the same π partition, the initial states of the corresponding β partitions are in the same θ partition.

We define ξ as follows: $\xi.i = \gamma(r(\pi\alpha.i))$.

We are now ready to prove SKS2. Let $s \in {}^{\pi}\sigma^{i}$. We show that $sR\delta(\xi.i)$. By the definition of π, we have

$$s \in {}^{\alpha}\sigma^{\pi\alpha.i} \vee \cdots \vee s \in {}^{\alpha}\sigma^{\pi\alpha(i+1)-1}$$

Hence,

$$sP\tau(\beta(\pi\alpha.i)) \vee \cdots \vee sP\tau(\beta(\pi\alpha(i+1)-1))$$

Note that by the definition of r (apply r to $\pi\alpha.i$):

$$\theta(r(\pi\alpha.i)) \leq \beta(\pi\alpha.i) < \theta(r(\pi\alpha.i)+1)$$

Hence,

$$\tau(\beta(\pi\alpha.i))Q\delta(\gamma(r(\pi\alpha.i))) \vee \cdots \vee \tau(\beta(\pi\alpha(i+1)-1))Q\delta(\gamma(r(\pi\alpha(i+1)-1)))$$

By the definition of ξ and the relationship between r and $\pi\alpha$ described above, we simplify the above formula as follows.

$$\tau(\beta(\pi\alpha.i))Q\delta(\xi.i) \vee \cdots \vee \tau(\beta(\pi\alpha(i+1)-1))Q\delta(\xi.i)$$

Therefore, by the definition of R, we have that $sR\delta(\xi.i)$ holds. □

Theorem 2. *The reflexive transitive closure of an SKS is an SKS.*

Theorem 3. *Given a TS $\mathcal{M}$, the greatest SKS on $\mathcal{M}$ is a preorder.*

Proof. Let G be the greatest SKS on $\mathcal{M}$. From Theorem 2, G^* is an SKS. Hence $G^* \subseteq G$. Furthermore, since $G \subseteq G^*$, we have that $G = G^*$, *i.e.*, G is reflexive and transitive. □

3.2 Skipping Refinement

We now recall the notion of skipping refinement [10]. We use skipping simulation, a notion defined in terms of a single transition system, to define skipping refinement, a notion that relates *two* transition systems: an *abstract* transition system and a *concrete* transition system. Informally, if a concrete system is a skipping refinement of an abstract system, then its observable behaviors are also behaviors of the abstract system, modulo skipping (which includes stuttering). The notion is parameterized by a *refinement map*, a function that maps concrete states to their corresponding abstract states. A refinement map along with a labeling function determines what is observable at a concrete state.

Definition 4 Skipping Refinement. *Let $\mathcal{M}_A = \langle S_A, \xrightarrow{A}, L_A\rangle$ and $\mathcal{M}_C = \langle S_C, \xrightarrow{C}, L_C\rangle$ be transition systems and let $r : S_C \to S_A$ be a refinement map. We say $\mathcal{M}_C$ is a skipping refinement of $\mathcal{M}_A$ with respect to r, written $\mathcal{M}_C \lesssim_r \mathcal{M}_A$, if there exists a binary relation B such that all of the following hold.*

1. $\langle \forall s \in S_C :: sBr.s \rangle$ and
2. B *is an SKS on* $\langle S_C \uplus S_A, \xrightarrow{C} \uplus \xrightarrow{A}, \mathcal{L} \rangle$ *where* $\mathcal{L}.s = L_A(s)$ *for* $s \in S_A$, *and* $\mathcal{L}.s = L_A(r.s)$ *for* $s \in S_C$.

Next, we use the property that skipping simulation is closed under relational composition to show that skipping refinement supports modular reasoning using a stepwise refinement approach. In order to verify that a low-level complex implementation $\mathcal{M}_C$ refines a simple high-level abstract specification $\mathcal{M}_A$ one proceeds as follows: starting with $\mathcal{M}_A$ define a sequence of intermediate systems leading to the final complex implementation $\mathcal{M}_C$. Any two successive systems in the sequence differ only in relatively few aspects of their behavior. We then show that, at each step in the sequence, the system at the current step is a refinement of the previous one. Since at each step, the verification effort is focused only on the few differences in behavior between two systems under consideration, proof obligations are simpler than the monolithic proof. Note that this methodology is orthogonal to (horizontal) modular reasoning that infers the correctness of a system from the correctness of its sub-components.

Theorem 4. *Let* $\mathcal{M}_1 = \langle S_1, \xrightarrow{1}, L_1 \rangle$, $\mathcal{M}_2 = \langle S_2, \xrightarrow{2}, L_2 \rangle$, *and* $\mathcal{M}_3 = \langle S_3, \xrightarrow{3}, L_3 \rangle$ *be TSs,* $p : S_1 \to S_2$ *and* $r : S_2 \to S_3$. *If* $\mathcal{M}_1 \lesssim_p \mathcal{M}_2$ *and* $\mathcal{M}_2 \lesssim_r \mathcal{M}_3$, *then* $\mathcal{M}_1 \lesssim_{p;r} \mathcal{M}_3$.

Proof: Since $\mathcal{M}_1 \lesssim_p \mathcal{M}_2$, we have an SKS, say A, such that $\langle \forall s \in S_1 :: sA(p.s) \rangle$. Furthermore, without loss of generality we can assume that $A \subseteq S_1 \times S_2$. Similarly, since $\mathcal{M}_2 \lesssim_r \mathcal{M}_3$, we have an SKS, say B, such that $\langle \forall s \in S_2 :: sB(r.s) \rangle$ and $B \subseteq S_2 \times S_3$. Define $C = A; B$. Then we have that $C \subseteq S_1 \times S_3$ and $\langle \forall s \in S_1 :: sCr(p.s) \rangle$. Also, from Theorem 2, C is an SKS on $\langle S_1 \uplus S_3, \xrightarrow{1} \uplus \xrightarrow{3}, \mathcal{L} \rangle$, where $\mathcal{L}.s = L_3(s)$ if $s \in S_3$ else $\mathcal{L}.s = L_3(r(p.s))$.

Formally, to establish that a complex low-level implementation $\mathcal{M}_C$ refines a simple high-level abstract specification $\mathcal{M}_A$, one defines intermediate systems $\mathcal{M}_1, \ldots \mathcal{M}_n$, where $n \geq 1$ and establishes the following: $\mathcal{M}_C = \mathcal{M}_0 \lesssim_{r_0} \mathcal{M}_1 \lesssim_{r_1} \ldots \lesssim_{r_{n-1}} \mathcal{M}_n = \mathcal{M}_A$. Then from Theorem 4, we have that $\mathcal{M}_C \lesssim_r \mathcal{M}_A$, where $r = r_0; r_1; \ldots; r_{n-1}$. We illustrate the utility of this approach in Sect. 5 by proving the correctness of an optimized event processing systems.

Theorem 5. *Let* $\mathcal{M} = \langle S, \to, L \rangle$ *be a TS. Let* $\mathcal{M}' = \langle S', \to', L' \rangle$ *where* $S' \subseteq S$, $\to' \subseteq S' \times S'$, $\to'$ *is a left-total subset of* $\to^+$, *and* $L' = L|_{S'}$. *Then* $\mathcal{M}' \lesssim_I \mathcal{M}$, *where* I *is the identity function on* S'.

Corollary 2. *Let* $\mathcal{M}_C = \langle S_C, \xrightarrow{C}, L_C \rangle$ *and* $\mathcal{M}_A = \langle S_A, \xrightarrow{A}, L_A \rangle$ *be TSs,* $r : S_C \to S_A$ *be a refinement map. Let* $\mathcal{M}'_C = \langle S'_C, \xrightarrow{C}{}', L'_C \rangle$ *where* $S'_C \subseteq S_C$, $\xrightarrow{C}{}'$ *is a left-total subset of* $\xrightarrow{C}{}^+$, *and* $L'_C = L_C|_{S'_C}$. *If* $\mathcal{M}_C \lesssim_r \mathcal{M}_A$ *then* $\mathcal{M}'_C \lesssim_{r'} \mathcal{M}_A$, *where* r' *is* $r|_{S'_C}$.

We now illustrate the usefulness of the theory of skipping refinement using our running example of event processing systems. Consider MPEPS, that uses

a priority queue to find a non-empty set of events (say E_t) scheduled to execute at the current time and executes them. We allow the priority queue in MPEPS to be deterministic or nondeterministic. For example, the priority queue may deterministically select a single event in E_t to execute, or based on considerations such as resource utilization it may execute some subset of events in E_t in a single step. When reasoning about the correctness of MPEPS, one thing to notice is that there is a difference in the data structures used in the two systems: MPEPS uses a priority queue to effectively find the next set of events to execute in the scheduler, while AEPS uses a simple abstract set representation for the scheduler. Another thing to notice is that MPEPS can "execute faster" than AEPS in two ways: it can increment time by more than 1 and it can execute more than one event in a single step. The theory of skipping refinement developed in this paper enables us to separate out these concerns and apply a stepwise refinement approach to effectively analyse MPEPS.

First, we account for the difference in the data structures between MPEPS and AEPS. Towards this we define an intermediate system MEPS that is identical to MPEPS except that the scheduler in MEPS is now represented as a set of event-time pairs. Under a refinement map, say p, that extracts the set of event-time pairs in the priority queue of MPEPS, a step in MPEPS can be matched by a step in MEPS. Hence, MPEPS $\lesssim_p$ MEPS. Next we account for the difference between MEPS and AEPS in the number of events the two systems may execute in a single step. Towards this, observe that the state space of MEPS and tEPS are equal and the transition relation of MEPS is a left-total subset of the transitive closure of the transition relation of tEPS. Hence, from Theorem 5, we infer that MPEPS is a skipping refinement of tEPS using the identity function, say I_1, as the refinement map, *i.e.*, MEPS $\lesssim_{I_1}$ tEPS. Next observe that the state spaces of tEPS and AEPS are equal and the transition relation of tEPS is a left-total subset of the transitive closure of the transition relation of AEPS. Hence, from Theorem 5, tEPS is a skipping refinement of AEPS using the identity function, say I_2, as the refinement map, *i.e.*, tEPS $\lesssim_{I_2}$ AEPS. Finally, from the transitivity of skipping refinement (Theorem 4), we conclude that MPEPS $\lesssim_{p'}$ AEPS, where $p' = p; I_1; I_2$.

4 Mechanised Reasoning

To prove that a transition system $\mathcal{M}_C$ is a skipping refinement of a transition system $\mathcal{M}_A$ using Definition 3, requires us to show that for any fullpath from $\mathcal{M}_C$ we can find a matching fullpath from $\mathcal{M}_A$. However, reasoning about existence of infinite sequences can be problematic using automated tools. In this section, we develop sound and complete local proof methods that are applicable even if a system exhibits unbounded skipping. We first briefly present the proof methods, reduced well-founded skipping and well-founded skipping simulation, developed in [10].

Definition 5 Reduced Well-founded Skipping [10]. *$B \subseteq S \times S$ is a reduced well-founded skipping relation on TS $\mathcal{M} = \langle S, \rightarrow, L \rangle$ iff:*

(RWFSK1) $\langle \forall s, w \in S : sBw : L.s = L.w \rangle$
(RWFSK2) *There exists a function,* $rankt : S \times S \to W$, *such that* $\langle W, \prec \rangle$ *is well-founded and*

$$\langle \forall s,u, w \in S : s \to u \wedge sBw :$$
$$(a)\ (uBw \wedge rankt(u, w) \prec rankt(s, w)) \vee$$
$$(b)\ \langle \exists v : w \to^{+} v : uBv \rangle \rangle$$

Definition 6 Well-founded Skipping [10]**.** *$B \subseteq S \times S$ is a well-founded skipping relation on TS $\mathcal{M} = \langle S, \to, L \rangle$ iff:*

(WFSK1) $\langle \forall s, w \in S : sBw : L.s = L.w \rangle$
(WFSK2) *There exist functions,* $rankt : S \times S \to W$, $rankl : S \times S \times S \to \omega$, *such that* $\langle W, \prec \rangle$ *is well-founded and*

$$\langle \forall s,u, w \in S : s \to u \wedge sBw :$$
$$(a)\ \langle \exists v : w \to v : uBv \rangle \vee$$
$$(b)\ (uBw \wedge rankt(u, w) \prec rankt(s, w)) \vee$$
$$(c)\ \langle \exists v : w \to v : sBv \wedge rankl(v, s, u) < rankl(w, s, u) \rangle \vee$$
$$(d)\ \langle \exists v : w \to^{\geq 2} v : uBv \rangle \rangle$$

Theorem 6 [10]. *Let $\mathcal{M} = \langle S, \to, L \rangle$ be a TS and $B \subseteq S \times S$. The following statements are equivalent*

(i) B is a SKS on $\mathcal{M}$;
(ii) B is a WFSK on $\mathcal{M}$;
(iii) B is a RWFSK on $\mathcal{M}$.

Recall the event processing systems AEPS and tEPS described in Sect. 1.1. When no events are scheduled to execute at a given time, say t, tEPS increments time t to the earliest time in future, say $k > t$, at which an event is scheduled for execution. Execution of an event can add an event that is scheduled to be executed at an arbitrary time in future. Therefore, we cannot apriori determine an upper-bound on k. Using any of the above two proof-methods to reason about skipping refinement would require unbounded reachability analysis (conditions RWFSK2b and WFSK2d), often difficult for automated verification tools. To redress the situation, we develop two new proof methods of SKS; both require only local reasoning about steps and their successors.

Definition 7 Reduced Local Well-founded Skipping. *$B \subseteq S \times S$ is a local well-founded skipping relation on TS $\mathcal{M} = \langle S, \to, L \rangle$ iff:*

(RLWFSK1) $\langle \forall s, w \in S : sBw : L.s = L.w \rangle$
(RLWFSK2) *There exist functions,* $rankt : S \times S \longrightarrow W$, $rankls : S \times S \longrightarrow \omega$ *such that* $\langle W, \prec \rangle$ *is well founded, and, a binary relation* $\mathcal{O} \subseteq S \times S$

such that

$\langle \forall s, u, w \in S : sBw \wedge s \rightarrow u :$
$\quad (a)\ (uBw \wedge rankt(u, w) \prec rankt(s, w)) \vee$
$\quad (b)\ \langle \exists v : w \rightarrow v : u \mathcal{O} v \rangle \rangle$
and
$\langle \forall x, y \in S : x \mathcal{O} y :$
$\quad (c)\ xBy \vee$
$\quad (d)\ \langle \exists z : y \rightarrow z : x \mathcal{O} z \wedge rankls(z, x) < rankls(y, x) \rangle \rangle$

Observe that to prove that a relation is an RLWFSK on a transition system, it is sufficient to reason about single steps of the transition system. Also, note that RLWFSK does not differentiate between skipping and stuttering on the right. This is based on an earlier observation that skipping subsumes stuttering. We used this observation to simplify the definition. However, it can often be useful to differentiate between skipping and stuttering. Next we define local well-founded skipping simulation (LWFSK), a characterization of skipping simulation that separates reasoning about skipping and stuttering on the right (Fig. 2).

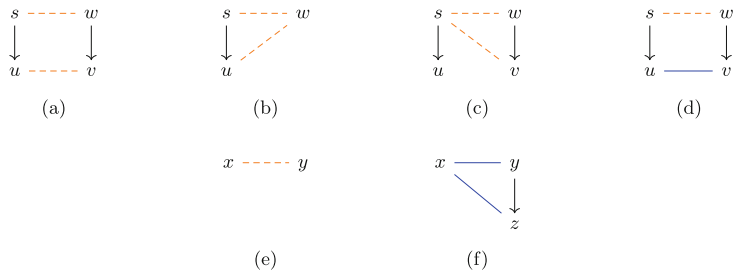


Fig. 2. Local well-founded skipping simulation (orange line indicates the states are related by B and blue line indicate the states are related by $\mathcal{O}$) (Color figure online)

Definition 8 Local Well-founded Skipping. *$B \subseteq S \times S$ is a local well-founded skipping relation on TS $\mathcal{M} = \langle S, \rightarrow, L \rangle$ iff:*

(LWFSK1) $\langle \forall s, w \in S : sBw : L.s = L.w \rangle$
(LWFSK2) *There exist functions, $rankt : S \times S \longrightarrow W$, $rankl : S \times S \times S \longrightarrow \omega$, and $rankls : S \times S \longrightarrow \omega$ such that $\langle W, \prec \rangle$ is well founded, and, a*

binary relation $\mathcal{O} \subseteq S \times S$ *such that*

$\langle \forall s, u, w \in S : sBw \wedge s \rightarrow u :$
(a) $\langle \exists v : w \rightarrow v : uBv \rangle \vee$
(b) $(uBw \wedge rankt(u, w) \prec rankt(s, w)) \vee$
(c) $\langle \exists v : w \rightarrow v : sBv \wedge rankl(v, s, u) < rankl(w, s, u) \rangle \vee$
(d) $\langle \exists v : w \rightarrow v : u\mathcal{O}v \rangle \rangle$
and
$\langle \forall x, y \in S : x\mathcal{O}y :$
(e) $xBy \vee$
(f) $\langle \exists z : y \rightarrow z : x\mathcal{O}z \wedge rankls(z, x) < rankls(y, x) \rangle \rangle$

Like RLWFSK, to prove that a relation is a LWFSK, reasoning about single steps of the transition system suffices. However, LWFSK2b accounts for stuttering on the right, and LWFSK2d along with LWFSK2e and LWFSK2f accounts for skipping on the right. Also observe that states related by $\mathcal{O}$ are not required to be labeled identically and may have no observable relationship to the states related by B.

Soundness and Completeness. We next show that RLWFSK and LWFSK in fact completely characterize skipping simulation, *i.e.*, RLWFSK and LWFSK are sound and complete proof rules. Thus if a concrete system $\mathcal{M}_C$ is a skipping refinement of $\mathcal{M}_A$, one can always effectively reason about it using RLWFSK and LWFSK.

Theorem 7. *Let* $\mathcal{M} = \langle S, \rightarrow, L \rangle$ *be a transition system and* $B \subseteq S \times S$*. The following statements are equivalent:*

(i) B *is an SKS on* $\mathcal{M}$*;*
(ii) B *is a WFSK on* $\mathcal{M}$*;*
(iii) B *is an RWFSK on* $\mathcal{M}$*;*
(iv) B *is an RLWFSK on* $\mathcal{M}$*;*
(v) B *is a LWFSK on* $\mathcal{M}$*;*

Proof: The equivalence of (i), (ii) and (iii) follows from Theorem 6. That (iv) implies (v) follows from the simple observation that RLWFSK2 implies LWFSK2. To complete the proof, we prove the following two implications. We prove below that (v) implies (ii) in Lemma 4 and that (iii) implies (iv) in Lemma 5. □

Lemma 4. *If* B *is a LWFSK on* $\mathcal{M}$*, then* B *is a WFSK on* $\mathcal{M}$*.*

Proof. Let B be a LWFSK on $\mathcal{M}$. WFSK1 follows directly from LWFSK1. Let *rankt*, *rankl*, and *rankls* be functions, and $\mathcal{O}$ be a binary relation such that LWFSK2 holds. To show that WFSK2 holds, we use the same *rankt* and *rankl* functions and let $s, u, w \in S$ and $s \rightarrow u$ and sBw. LWFSK2a, LWFSK2b and

LWFSK2c are equivalent to WFSK2a, WFSK2b and WFSK2c, respectively, so we show that if only LWFSK2d holds, then WFSK2d holds. Since LWFSK2d holds, there is a successor v of w such that $u\mathcal{O}v$. Since $u\mathcal{O}v$ holds, either LWFSK2e or LWFSK2f must hold between u and v. However, since LWFSK2a does not hold, LWFSK2e cannot hold and LWFSK2f must hold, *i.e.*, there exists a successor v' of v such that $u\mathcal{O}v' \wedge rankls(v', u) < rankls(v, u)$. So, we need a path of at least 2 steps from w to satisfy the universally quantified constraint on $\mathcal{O}$. Let us consider an arbitrary path, δ, such that $\delta.0 = w$, $\delta.1 = v$, $\delta.2 = v'$, $u\mathcal{O}\delta.i$, LWFSK2e does not hold between u and $\delta.i$ for $i \geq 1$, and $rankls(\delta.(i+1), u) < rankls(\delta.i, u)$. Notice that any such path must be finite because $rankls$ is well founded. Hence, δ is a finite path and there exists a $k \geq 2$ such that LWFSK2e holds between u and $\delta.k$. Therefore, WFSK2d holds, *i.e.*, there is a state in δ reachable from w in two or more steps which is related to u by B. □

Lemma 5. *If B is RWFSK on $\mathcal{M}$, then B is an RLWFSK on $\mathcal{M}$.*

Proof. Let B be an RWFSK on $\mathcal{M}$. RLWFSK1 follows directly from RWFSK1. To show that RLWFSK2 holds, we use any *rankt* function that can be used to show that RWFSK2 holds. We define $\mathcal{O}$ as follows.

$$\mathcal{O} = \{(u, v) : \langle \exists z : v \rightarrow^{+} z : uBz \rangle\}$$

We define $rankls(u, v)$ to be the minimal length of a $\mathcal{M}$-segment that starts at v and ends at a state, say z, such that uBz, if such a segment exists and 0 otherwise. Let $s, u, w \in S$, sBw and $s \rightarrow u$. If RWFSK2a holds between s, u, and w, then RLWFSK2a also holds. Next, suppose that RWFSK2a does not hold but RWFSK2b holds, *i.e.*, there is an $\mathcal{M}$-segment $\langle w, a, \ldots, v \rangle$ such that uBv; therefore, $u\mathcal{O}a$ and RLWFSK2b holds.

To finish the proof, we show that $\mathcal{O}$ and $rankls$ satisfy the constraints imposed by the second conjunct in RLWFSK2. Let $x, y \in S$, $x\mathcal{O}y$ and $x \not\mathcal{B} y$. From the definition of $\mathcal{O}$, we have that there is an $\mathcal{M}$-segment from y to a state related to x by B; let $\vec{y}$ be such a segment of minimal length. From definition of $rankls$, we have $rankls(y, x) = |\vec{y}|$. Observe that y cannot be the last state of $\vec{y}$ and $|\vec{y}| \geq 2$. This is because the last state in $\vec{y}$ must be related to x by B, but from the assumption we know that $x \not\mathcal{B} y$. Let y' be a successor of y in $\vec{y}$. Clearly, $x\mathcal{O}y'$; therefore, $rankls(y', x) < |\vec{y}| - 1$, since the length of a minimal $\mathcal{M}$-segment from y' to a state related to x by B, must be less or equal to $|\vec{y}| - 1$. □

5 Case Study (Event Processing System)

In this section, we analyze the correctness of an optimized event processing system (PEPS) that uses a *priority queue* to find an event scheduled to execute at any given time. We show that PEPS refines AEPS, a simple event processing system described in Sect. 1. Our goal is to illustrate the benefits of the theory of skipping refinement and the associated local proof methods developed in the

paper. We use ACL2s [3], an interactive theorem prover, to define the operational semantics of the systems and mechanize a proof of its correctness.

Operational Semantics of PEPS: A state of PEPS system is a triple $\langle$`tm`, `otevs`, `mem`$\rangle$, where `tm` is a natural number denoting current time, `otevs` is a set of timed-event pairs denoting the scheduler that is ordered with respect to a total order `te-<` on timed-event pairs, and `mem` is a collection of variable-integer pairs denoting the shared memory. The transition function of PEPS is defined as follows: if there are no events in `otevs`, then PEPS just increments the current time by 1. Otherwise, it picks the first timed-event pair, say $\langle e, t\rangle$ in `otevs`, executes it and updates the time to t. The execution of an event may result in adding new timed-events to the scheduler, removing existing timed-events from the scheduler and updating the memory. Finally, the executed timed-event is removed from the scheduler. This is a simple, generic model of an event processing system. Notice that the ability to remove events can be used to specify systems with preemption [23]: an event scheduled to execute at some future time may be canceled (and possibly rescheduled to be executed at a different time in future) as a result of the execution of an event that preempts it. Notice that, for a given total order, PEPS is a deterministic system.

The execution of an event is modeled using three constrained functions that take as input an event, `ev`, a time, `t`, and a memory, `mem`: `step-events-add` returns the set of new timed-event pairs to add to the scheduler; `step-events-rm` returns the set of timed-event pairs to remove from the scheduler; and `step-memory` returns a memory updated as specified by the event. We place minimal constraints on these functions. For example, we only require that `step-events-add` returns a set of event-time pairs of the form $\langle e, t_e\rangle$ where t_e is greater than the current time t. The constrained functions are defined using the `encapsulate` construct in ACL2 and can be instantiated with any executable definitions that satisfy these constraints without affecting the proof of correctness of PEPS. Moreover, note that the particular choice of the total order on timed-event pairs is irrelevant to the proof of correctness of PEPS.

Stepwise Refinement: We show that PEPS refines AEPS using a stepwise refinement approach: first we define an intermediate system HPEPS obtained by augmenting PEPS with history information and show that PEPS is a simulation refinement of HPEPS. Second, we show that HPEPS is a skipping refinement of AEPS. Finally, we appeal to Theorems 1 and 4 to infer that PEPS refines AEPS. Note that the compositionality of skipping refinement enables us to decompose the proof into a sequence of refinement proofs, each of which is simpler. Moreover, the history information in HPEPS is helpful in defining the witnessing binary relation and the rank function required to prove skipping refinement.

An HPEPS state is a four-tuple $\langle tm, otevs, mem, h\rangle$, where `tm`, `otevs`, `mem` are respectively the current time, an ordered set of timed events and a collection of variable-integer pairs, and `h` is the history information. The history information `h` consists of a Boolean variable `valid`, time `tm`, and an ordered set of timed-event pairs `otevs` and the memory `mem`. Intuitively, `h` records the state preceding the

current state. The transition function HPEPS is same as the transition function of PEPS except that HPEPS also records the history in `h`.

PEPS Refines HPEPS: Observe that, modulo the history information, a step of PEPS directly corresponds to a step of HPEPS, *i.e.*, PEPS is a bisimulation refinement of HPEPS under a refinement map that projects a PEPS state $\langle tm, otevs, mem\rangle$ to the HPEPS state $\langle tm, otevs, mem, h\rangle$ where the `valid` component of `h` is set to false. But we only prove that it is a simulation refinement, because, from Theorem 1, it suffices to establish that PEPS is a skipping refinement of HPEPS. The proofs primarily require showing that two sets of ordered timed-events that are set equivalent are in fact equal and that adding and removing equivalent sets of timed-event from equal schedulers results in equal schedulers.

HPEPS Refines AEPS: Next we show that HPEPS is a skipping refinement of AEPS under the refinement map R, a function that simply projects an HPEPS state to an AEPS state. To show that HPEPS is a skipping refinement of AEPS under the refinement map R, from Definition 4, we must show as witness a binary relation B that satisfies the two conditions. Let $B = \{(s, R.s) : s \textit{ is an HPEPS state}\}$. To establish that B is an SKS on the disjoint union of HPEPS and AEPS, we have a choice of four proof-methods (Sect. 4). Recall that execution of an event can add a new event scheduled to be executed at an arbitrary time in the future. As a result, if we were to use WFSK or RWFSK, the proof obligations from conditions WFSK2d (Definition 5) and RWFSK2b (Definition 6) would require unbounded reachability analysis, something that typically places a big burden on verification tools and their users. In contrast, the proof obligations to establish RLWFSK are local and only require reasoning about states and their successors, which significantly reduces the proof complexity.

RLWFSK1 holds trivially. To prove that RLWFSK2 holds we define a binary relation $\mathcal{O}$ and a rank function *rankls* and show that they satisfy the two universally quantified formulas in RLWFSK2. Moreover, since HPEPS does not stutter we ignore RLWFSK2a, and that is why we do not define *rankt*. Finally, our proof obligation is: for all HPEPS s, u and AEPS state w such that $s \rightarrow u$ and sBw holds, there exists a AEPS state v such that $w \rightarrow v$ and $u\mathcal{O}v$ holds.

Verification Effort: We used the `defdata` framework in ACL2s, to specify the data definitions for the three systems and the `definec` construct to introduce function definitions along with their input-contracts (pre-conditions) and output-contracts (post-conditions). In addition to admitting a data definition, `defdata` proves several theorems about the functions that are extremely helpful in automatically discharging type-like proof obligations. We also developed a library to concisely describe functions using higher-order constructs like map and reduce, which made some of the definitions clearer. ACL2s supports first-order quantifiers via the `defun-sk` construct, which essentially amounts to the use of Hilbert's choice operator. We use `defun-sk` to model the transition relation for AEPS (a non-deterministic system) and to specify the proof obligations for proving that HPEPS refines AEPS. However, support for automated reasoning

about quantifiers is limited in ACL2. Therefore, we use the domain knowledge, when possible (*e.g.*, a system is deterministic), to eliminate quantifiers in the proof obligations and provide explicit witnesses for existential quantifiers.

The proof makes essential use of several libraries available in ACL2 for reasoning about lists and sets. In addition, we prove a collection of additional lemmas that can be roughly categorized into four categories. First, we have a collection of lemmas to prove the input-output contracts of the functions. Second, we have a collection of lemmas to show that operations on the schedulers in the three systems preserve various invariants, *e.g.*, that any timed-event in the scheduler is scheduled to execute at a time greater or equal to the current time. Third, we have a collection of lemmas to show that inserting and removing two equivalent sets of timed-events from a scheduler results in an equivalent scheduler. And fourth, we have a collection of lemmas to show that two schedulers are equivalent *iff* they are set equal. The above lemmas are used to establish a relationship between priority queues, a data structure used by the implementation system, and sets, the corresponding data structure used in the specification system. The behavioral difference between the two systems is accounted for by the notion of skipping refinement. This separation significantly eases understanding as well as mechanical reasoning about the correctness of reactive systems. We have 8 top-level proof obligations and a few dozen supporting lemmas. The entire proof takes about 120 s on a machine with 2.2 GHz Intel Core i7 with 16 GB main memory.

6 Related Work

Several notions of correctness have been proposed in the literature and their properties been widely studied [2,5,11,16,17]. In this paper, we develop a theory of skipping refinement to effectively prove the correctness of optimized reactive systems using automated verification tools. These results establish skipping refinement on par with notions of refinement based on (bi)simulation [22] and stuttering (bi)simulation [20,24], in the sense that skipping refinement is (1) compositional and (2) admits local proofs methods. Together the two properties have been instrumental in significantly reducing the proof complexity in verification of large and complex systems. We developed the theory of skipping refinement using a generic model of transition systems and place no restrictions on the state space size or the branching factor of the transition system. Any system with a well-defined operational semantics can be mapped to a labeled transition system. Moreover, the local proof methods are sound and complete, *i.e.*, if an implementation is a skipping refinement of the specification, we can always use the local proof methods to effectively reason about it.

Refinement-based methodologies have been successfully used to verify the correctness of several realistic hardware and software systems. In [13], several complex concurrent programs were verified using a stepwise refinement methodology. In addition, Kragl and Qadeer [13] also develop a compact representation to facilitate the description of programs at different levels of abstraction and associated refinement proofs. Several back-end compiler transformations are proved

correct in Compcert [15] using simulation refinement. In [25], several compiler transformations were verified using stuttering refinement and associated local proof methods. Recently, refinement-based methodology has also been applied to verify the correctness of practical distributed systems [8] and a general-purpose operating system microkernel [12]. The full verification of CertiKOS [6,7], an OS kernel, is based on the notion of simulation refinement. Refinement based approaches have also been extensively used to verify microprocessor designs [1,9,19,21,26]. Skipping refinement was used to verify the correctness of optimized memory controllers and a JVM-inspired stack machine [10].

7 Conclusion and Future Work

In this paper, we developed the theory of skipping refinement. Skipping refinement is designed to reason about the correctness of optimized reactive systems, a class of systems where a single transition in a concrete low-level implementation may correspond to a sequence of observable steps in the corresponding abstract high-level specification. Examples of such systems include optimizing compilers, concurrent and parallel systems and superscalar processors. We developed sound and complete proof methods that reduce global reasoning about infinite computations of such systems to local reasoning about states and their successors. We also showed that the skipping simulation is closed under composition and therefore is amenable to modular reasoning using a stepwise refinement approach. We experimentally validated our results by analyzing the correctness of an optimized event-processing system in ACL2s. For future work, we plan to precisely classify temporal logic properties that are preserved by skipping refinement. This would enable us to transfer temporal properties from specifications to implementations, after establishing refinement.

References

1. Aagaard, M.D., Cook, B., Day, N.A., Jones, R.B.: A framework for microprocessor correctness statements. In: Margaria, T., Melham, T. (eds.) CHARME 2001. LNCS, vol. 2144, pp. 433–448. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44798-9_33
2. Basten, T.: Branching bisimilarity is an equivalence indeed!. Inf. Process. Lett. **58**, 141–147 (1996)
3. Chamarthi, H.R., Dillinger, P., Manolios, P., Vroon, D.: The ACL2 sedan theorem proving system. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 291–295. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_27
4. Clarke, E.M., Grumberg, O., Browne, M.C.: Reasoning about networks with many identical finite-state processes. In: PODC (1986)
5. van Glabbeek, R.J.: The linear time-branching time spectrum (extended abstract). In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 278–297. Springer, Heidelberg (1990). https://doi.org/10.1007/BFb0039066

6. Gu, L., Vaynberg, A., Ford, B., Shao, Z., Costanzo, D.: CertiKOS: a certified kernel for secure cloud computing. In: APSys (2011)
7. Gu, R., et al.: Deep specifications and certified abstraction layers. In: POPL (2015)
8. Hawblitzel, C., et al.: IronFleet: Proving practical distributed systems correct. In: SOSP (2015)
9. Hosabettu, R., Gopalakrishnan, G., Srivas, M.: Formal verification of a complex pipelined processor. Form. Methods Syst. Des. **23**, 171–213 (2003)
10. Jain, M., Manolios, P.: Skipping refinement. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 103–119. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_7
11. Klarlund, N.: Progress measures and finite arguments for infinite computations. Ph.D. thesis (1990)
12. Klein, G., Sewell, T., Winwood, S.: Refinement in the formal verification of the seL4 microkernel. In: Hardin, D. (ed.) Design and Verification of Microprocessor Systems for High-Assurance Applications, pp. 323–339. Springer, Boston (2010). https://doi.org/10.1007/978-1-4419-1539-9_11
13. Kragl, B., Qadeer, S.: Layered concurrent programs. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 79–102. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_5
14. Lamport, L.: What good is temporal logic. Information processing (1993)
15. Leroy, X., Blazy, S.: Formal verification of a c-like memory model and its uses for verifying program transformations. J. Autom. Reason. **41**, 1–31 (2008)
16. Liu, X., Yu, T., Zhang, W.: Analyzing divergence in bisimulation semantics. In: POPL (2017)
17. Lynch, N.A., Vaandrager, F.W.: Forward and backward simulations: I. Untimed systems. Inf. Comput. (1995)
18. Manolios, P.: Mechanical verification of reactive systems. Ph.D. thesis, University of Texas (2001)
19. Manolios, P.: Correctness of pipelined machines. In: Hunt, W.A., Johnson, S.D. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 181–198. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-40922-X_11
20. Manolios, P.: A compositional theory of refinement for branching time. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 304–318. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39724-3_28
21. Manolios, P., Srinivasan, S.K.: A complete compositional reasoning framework for the efficient verification of pipelined machines. In: ICCAD (2005)
22. Milner, R.: An algebraic definition of simulation between programs. In: Proceedings of the 2nd International Joint Conference on Artificial Intelligence (1971)
23. Misra, J.: Distributed discrete-event simulation. ACM Comput. Surv. **18**, 39–65 (1986)
24. Namjoshi, K.S.: A simple characterization of stuttering bisimulation. In: Ramesh, S., Sivakumar, G. (eds.) FSTTCS 1997. LNCS, vol. 1346, pp. 284–296. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0058037
25. Namjoshi, K.S., Zuck, L.D.: Witnessing program transformations. In: Logozzo, F., Fähndrich, M. (eds.) SAS 2013. LNCS, vol. 7935, pp. 304–323. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38856-9_17
26. Ray, S., Jr Hunt, W.A.: Deductive verification of pipelined machines using first-order quantification. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 31–43. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_3

Robust Controller Synthesis in Timed Büchi Automata: A Symbolic Approach

Damien Busatto-Gaston[1(✉)], Benjamin Monmege[1], Pierre-Alain Reynier[1], and Ocan Sankur[2]

[1] Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{damien.busatto,pierre-alain.reynier}@lis-lab.fr, benjamin.monmege@univ-amu.fr
[2] Univ Rennes, Inria, CNRS, IRISA, Rennes, France
ocan.sankur@irisa.fr

Abstract. We solve in a purely symbolic way the robust controller synthesis problem in timed automata with Büchi acceptance conditions. The goal of the controller is to play according to an accepting lasso of the automaton, while resisting to timing perturbations chosen by a competing environment. The problem was previously shown to be PSPACE-complete using regions-based techniques, but we provide a first tool solving the problem using zones only, thus more resilient to state-space explosion problem. The key ingredient is the introduction of branching constraint graphs allowing to decide in polynomial time whether a given lasso is robust, and even compute the largest admissible perturbation if it is. We also make an original use of constraint graphs in this context in order to test the inclusion of timed reachability relations, crucial for the termination criterion of our algorithm. Our techniques are illustrated using a case study on the regulation of a train network.

1 Introduction

Timed automata [1] extend finite-state automata with timing constraints, providing an automata-theoretic framework to design, model, verify and synthesise real-time systems. However, the semantics of timed automata is a mathematical idealisation: it assumes that clocks have infinite precision and instantaneous actions. Proving that a timed automaton satisfies a property does not ensure that a real implementation of it also does. This *robustness* issue is a challenging problem for embedded systems [12], and alternative semantics have been proposed, so as to ensure that the verified (or synthesised) behaviour remains correct in presence of small timing perturbations.

We are interested in a fundamental controller synthesis problem in timed automata equipped with a Büchi acceptance condition: it consists in determining whether there exists an accepting infinite execution.

This work was funded by ANR project Ticktac (ANR-18-CE40-0015).

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 572–590, 2019.
https://doi.org/10.1007/978-3-030-25540-4_33

Thus, the role of the controller is to choose transitions and delays. This problem has been studied numerously in the exact setting [13–15,17,19,27,28]. In the context of robustness, this strategy should be tolerant to small perturbations of the delays. This discards strategies suffering from weaknesses such as Zeno behaviours, or even non-Zeno behaviours requiring infinite precision, as exhibited in [6].

More formally, the semantics we consider is defined as a game that depends on some parameter δ representing an upper bound on the amplitude of the perturbation [7]. In this game, the controller plays against an antagonistic environment that can perturb each delay using a value chosen in the interval $[-\delta, \delta]$. The case of a fixed value of δ has been shown to be decidable in [7], and also for a related model in [18]. However, these algorithms are based on regions, and as the value of δ may be very different from the constants appearing in the guards of the automaton, do not yield practical algorithms. Moreover, the maximal perturbation is not necessarily known in advance, and could be considered as part of the design process.

The problem we are interested in is *qualitative*: we want to determine whether *there exists* a positive value of δ such that the controller wins the game. It has been proven in [25] that this problem is in PSPACE (and even PSPACE-complete), thus no harder than in the exact setting with no perturbation allowed [1]. However, the algorithm heavily relies on regions, and more precisely on an abstraction that refines the one of regions, namely folded orbit graphs. Hence, it is not at all amenable to implementation.

Our objective is to provide an efficient symbolic algorithm for solving this problem. To this end, we target the use of *zones* instead of regions, as they allow an on-demand partitioning of the state space. Moreover, the algorithm we develop explores the reachable state-space in a *forward* manner. This is known to lead to better performances, as witnessed by the successful tool UPPAAL TIGA based on forward algorithms for solving controller synthesis problems [5].

Our algorithm can be understood as an adaptation to the robustness setting of the standard algorithm for Büchi acceptance in timed automata [17]. This algorithm looks for an accepting lasso using a double depth-first search. A major difficulty consists in checking whether a lasso can be robustly iterated, i.e. whether there exists $\delta > 0$ such that the controller can follow the cycle for an infinite amount of steps while being tolerant to perturbations of amplitude at most δ. The key argument of [25] was the notion of aperiodic folded orbit graph of a path in the region automaton, thus tightly connected to regions. Lifting this notion to zones seems impossible as it makes an important use of the fact that valuations in regions are time-abstract bisimilar, which is not the case for zones.

Our contributions are threefold. First, we provide a polynomial time procedure to decide, given a lasso, whether it can be robustly iterated. This symbolic algorithm relies on a computation of the greatest fixpoint of the operator describing the set of controllable predecessors of a path. In order to provide an argument of termination for this computation, we resort to a new notion of branching constraint graphs, extending the approach used in [16,26] and based

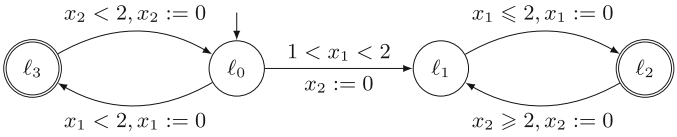


Fig. 1. A timed automaton

on constraint graphs (introduced in [8]) to check iterability of a cycle, without robustness requirements. Second, we show that when considering a lasso, not only can we decide robust iterability, but we can even compute the largest perturbation under which it is controllable. This problem was not known to be decidable before. Finally, we provide a termination criterion for the analysis of lassos. Focusing on zones is not complete: it can be the case that two cycles lead to the same zones, but one is robustly iterable while the other one is not. Robust iterability crucially depends on the real-time dynamics of the cycle and we prove that it actually only depends on the reachability relation of the path. We provide a polynomial-time algorithm for checking inclusion between reachability relations of paths in timed automata based on constraint graphs. It is worth noticing that all our procedures can be implemented using difference bound matrices, a very efficient data structure used for timed systems. These developments have been integrated in a tool, and we present a case study of a train regulation network illustrating its performances.

Integrating the robustness question in the verification of real-time systems has attracted attention in the community, and the recent works include, for instance, robust model checking for timed automata under clock drifts [23], Lipschitz robustness notions for timed systems [11], quantitative robust synthesis for timed automata [2]. Stability analysis and synthesis of stabilizing controllers in hybrid systems are a closely related topic, see e.g. [20,21].

2 Timed Automata: Reachability and Robustness

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a finite set of clock variables. It is extended with a virtual clock x_0, constantly equal to 0, and we denote by $\mathcal{X}_0$ the set $\mathcal{X} \cup \{x_0\}$. An atomic clock constraint on $\mathcal{X}$ is a formula $x - y \leqslant k$, or $x - y < k$ with $x \neq y \in \mathcal{X}_0$ and $k \in \mathbb{Q}$. A constraint is non-diagonal if one of the two clocks is x_0. We denote by $\mathsf{Guards}(X)$ (respectively, $\mathsf{Guards}_{\mathrm{nd}}(X)$) the set of (clock) constraints (respectively, non-diagonal clock constraints) built as conjunctions of atomic clock constraints (respectively, non-diagonal atomic clock constraints).

A clock valuation ν is an element of $\mathbb{R}_{\geqslant 0}^{\mathcal{X}}$. It is extended to $\mathbb{R}_{\geqslant 0}^{\mathcal{X}_0}$ by letting $\nu(x_0) = 0$. For all $d \in \mathbb{R}_{>0}$, we let $\nu + d$ be the valuation defined by $(\nu + d)(x) = \nu(x) + d$ for all clocks $x \in \mathcal{X}$. If $\mathcal{Y} \subseteq \mathcal{X}$, we also let $\nu[\mathcal{Y} \leftarrow 0]$ be the valuation resetting clocks in $\mathcal{Y}$ to 0, without modifying values of other clocks. A valuation ν satisfies an atomic clock constraint $x - y \bowtie k$ (with $\bowtie \in \{\leqslant, <\}$) if $\nu(x) - \nu(y) \bowtie k$. The satisfaction relation is then extended to clock constraints

naturally: the satisfaction of constraint g by a valuation ν is denoted by $\nu \models g$. The set of valuations satisfying a constraint g is denoted by $[\![g]\!]$.

A *timed automaton* is a tuple $\mathcal{A} = (L, \ell_0, E, L_t)$ with L a finite set of locations, $\ell_0 \in L$ an initial location, $E \subseteq L \times \mathsf{Guards}_{\mathrm{nd}}(\mathcal{X}) \times 2^{\mathcal{X}} \times L$ is a finite set of edges, and L_t is a set of accepting locations.

An example of timed automaton is depicted in Fig. 1, where the reset of a clock x is denoted by $x := 0$. The semantics of the timed automaton $\mathcal{A}$ is defined as an infinite transition system $[\![\mathcal{A}]\!] = (S, s_0, \to)$. The set S of states of $[\![\mathcal{A}]\!]$ is $L \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$, $s_0 = (\ell_0, \mathbf{0})$. A transition of $[\![\mathcal{A}]\!]$ is of the form $(\ell, \nu) \xrightarrow{e,d} (\ell', \nu')$ with $e = (\ell, g, \mathcal{Y}, \ell') \in E$ and $d \in \mathbb{R}_{>0}$ such that $\nu + d \models g$ and $\nu' = (\nu + d)[\mathcal{Y} \leftarrow 0]$. We call *path* a possible finite sequence of edges in the timed automaton. The *reachability relation* of a path ρ, denoted by $\mathsf{Reach}(\rho)$ is the set of pairs (ν, ν') such that there is a sequence of transitions of $[\![\mathcal{A}]\!]$ starting from (ℓ, ν), ending in (ℓ', ν') and that follows ρ in order as the edges of the timed automaton. A *run* of $\mathcal{A}$ is an infinite sequence of transitions of $[\![\mathcal{A}]\!]$ starting from s_0. We are interested in Büchi objectives. Therefore, a run is accepting if there exists a final location $\ell_t \in L_t$ that the run visits infinitely often.

As done classically, we assume that every clock is bounded in $\mathcal{A}$ by a constant M, that is we only consider the previous infinite transition system over the subset $L \times [0, M]^{\mathcal{X}}$ of states.

We study the robustness problem introduced in [25], that is stated in terms of games where a controller fights against an environment. After a prefix of a run, the controller will have the capability to choose delays and transitions to fire, whereas the environment perturbs the delays chosen by the controller with a small parameter $\delta > 0$. The aim of the controller will be to find a strategy so that, no matter how the environment plays, he is ensured to generate an infinite run satisfying the Büchi condition. Formally, given a timed automaton $\mathcal{A} = (L, \ell_0, E, L_t)$ and $\delta > 0$, the perturbation game is a two-player turn-based game $\mathcal{G}_\delta(\mathcal{A})$ between a controller and an environment. Its state space is partitioned into $S_C \uplus S_E$ where $S_C = L \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ belongs to the controller, and $S_E = L \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0} \times \mathbb{R}_{>0} \times E$ to the environment. The initial state is $(\ell_0, \mathbf{0}) \in S_C$. From each state $(\ell, \nu) \in S_C$, there is a transition to $(\ell, \nu, d, e) \in S_E$ with $e = (\ell, g, \mathcal{Y}, \ell') \in E$ whenever $d > \delta$, and $\nu + d + \varepsilon \models g$ for all $\varepsilon \in [-\delta, \delta]$. Then, from each state $(\ell, \nu, d, (\ell, g, \mathcal{Y}, \ell')) \in S_E$, there is a transition to $(\ell', (\nu + d + \varepsilon)[r \leftarrow 0]) \in S_C$ for all $\varepsilon \in [-\delta, \delta]$. A play of $\mathcal{G}_\delta(\mathcal{A})$ is a finite or infinite path $q_0 \xrightarrow{t_1} q_1 \xrightarrow{t_2} q_2 \cdots$ where $q_0 = (\ell_0, 0)$ and t_i is a transition from state q_{i-1} to q_i, for all $i > 0$. It is said to be maximal if it is infinite or can not be extended with any transition.

A strategy for the controller is a function σ_{Con} mapping each non-maximal play ending in some $(\ell, \nu) \in S_C$ to a pair (d, e) where $d > 0$ and $e \in E$ such that there is a transition from (ℓ, ν) to (ℓ, ν, d, e). A strategy for the environment is a function σ_{Env} mapping each finite play ending in (ℓ, ν, d, e) to a state (ℓ', ν') related by a transition. A play gives rise to a unique run of $[\![\mathcal{A}]\!]$ by only keeping states in V_C. For a pair of strategies $(\sigma_{\mathsf{Con}}, \sigma_{\mathsf{Env}})$, we let $\mathsf{play}^{\delta}_{\mathcal{A}}(\sigma_{\mathsf{Con}}, \sigma_{\mathsf{Env}})$ denote the run associated with the unique maximal play of $\mathcal{G}_\delta(\mathcal{A})$ that follows the strategies. Controller's strategy σ_{Con} is winning (with respect to the Büchi

objective L_t) if for all strategies σ_{Env} of the environment, $\mathsf{play}^{\delta}_{\mathcal{A}}(\sigma_{\mathsf{Con}}, \sigma_{\mathsf{Env}})$ is infinite and visits infinitely often some location of L_t. The *parametrised robust controller synthesis problem* asks, given a timed automaton $\mathcal{A}$, whether there exists $\delta > 0$ such that the controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$.

Example 1. The controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$, with $\mathcal{A}$ the automaton of Fig. 1, for all possible values of $\delta < 1/2$. Indeed, he can follow the cycle $\ell_0 \to \ell_3 \to \ell_0$ by always picking time delay $1/2$ so that, when arriving in ℓ_3 (resp. ℓ_0) after the perturbation of the environment, clock x_2 (resp. x_1) has a valuation in $[1/2-\delta, 1/2+\delta]$. Therefore, he can play forever following this memoryless strategy. For $\delta \geq 1/2$, the environment can enforce reaching ℓ_3 with a value for x_2 at least equal to 1. The guard $x_2 < 2$ of the next transition to ℓ_0 cannot be guaranteed, and therefore the controller cannot win $\mathcal{G}_\delta(\mathcal{A})$. In [25], it is shown that the cycle around ℓ_2 does not provide a winning strategy for the controller for any value of $\delta > 0$ since perturbations accumulate so that the controller can only play it a finite number of times in the worst case.

By [25], the parametrised robust controller synthesis problem is known to be PSPACE-complete. Their solution is based on the region automaton of $\mathcal{A}$. We are seeking for a more practical solution using zones. A zone Z over $\mathcal{X}$ is a convex subset of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ defined as the set of valuations satisfying a clock constraint g, i.e. $Z = [\![g]\!]$. Zones can be encoded into *difference-bound matrices (DBM)*, that are $|\mathcal{X}_0| \times |\mathcal{X}_0|$-matrices over $(\mathbb{R} \times \{<, \leqslant\}) \cup \{(\infty, <)\}$. We adopt the following notation: for a DBM M, we write $M = (\mathsf{M}, \prec^M)$, where M is the matrix made of the first components, with elements in $\mathbb{R} \cup \{\infty\}$, while $\prec^M$ is the matrix of the second components, with elements in $\{<, \leqslant\}$. A DBM M naturally represents a zone (which we abusively write M as well), defined as the set of valuations ν such that, for all $x, y \in \mathcal{X}_0$, $\nu(x) - \nu(y) \prec^M_{x,y} \mathsf{M}_{x,y}$ (where $\nu(x_0) = 0$). Coefficients of a DBM are thus pairs $(\prec, c)$. As usual, these can be compared: $(\prec, c)$ is less than $(\prec', c')$ (denoted by $(\prec, c) < (\prec', c')$) whenever $c < c'$ or ($c = c'$, $\prec = <$ and $\prec' = \leqslant$). Moreover, these coefficients can be added: $(\prec, c) + (\prec', c')$ is the pair $(\prec'', c + c')$ with $\prec'' = \leqslant$ if $\prec = \prec' = \leqslant$ and $\prec'' = <$ otherwise.

DBMs were introduced in [4,10] for analyzing timed automata; we refer to [3] for details. Standard operations used to explore the state space of timed automata have been defined on DBMs: intersection is written $M \cap N$, $\mathsf{Pretime}_{>t}(M)$ is the set of valuations such that a time delay of more than t time units leads to the zone M, $\mathsf{Unreset}_R(M)$ is the set of valuations that end in M when the clocks in R are reset. From a robustness perspective, we also consider the operator $\mathsf{shrink}_{[-\delta,\delta]}(M)$ defined as the set of valuations ν such that $\nu + [-\delta, \delta] \subseteq M$ introduced in [24]. Given a DBM M and a rational number δ, all these operations can be effectively computed in time cubic in $|\mathcal{X}|$.

3 Reachability Relation of a Path

Before treating the robustness issues, we start by designing a symbolic (i.e. zone-based) approach to describe and compare the reachability relations of paths

in timed automata. This will be crucial subsequently to design a termination criterion in the state space exploration of our robustness-checking algorithm. Solving the inclusion of reachability relations in a symbolic manner has independent interest and can have other applications.

The reachability relation $\mathsf{Reach}(\rho)$ of a path ρ, is a subset of $\mathbb{R}_{\geqslant 0}^{\mathcal{X}\cup\mathcal{X}'}$ where $\mathcal{X}'$ are primed versions of the clocks, such that each $(\nu,\nu') \in \mathsf{Reach}(\rho)$ iff there is a run from valuation ν to valuation ν' following ρ. Unfortunately, reachability relations $\mathsf{Reach}(\rho)$ are not zones in general, that is, they cannot be represented using only difference constraints. In fact, we shall see shortly that constraints of the form $x - y + z - u \leqslant c$ also appear, as already observed in [22]. We thus cannot rely directly on the traditional difference bound matrices (DBMs) used to represent zones. We instead rely on the constraint graphs that were introduced in [8], and explored in [16] for the parametric case (the latter work considers enlarged constraints, and not shrunk ones as we study here). Our contribution is to use these graphs to obtain a syntactic check of inclusion of the according reachability relations.

Constraint Graphs. Rather than considering the values of the clocks in $\mathcal{X}$, this data structure considers the date X_i of the latest reset of the clock x_i, and uses a new variable τ denoting the global timestamp. Note that the clock values can be recovered easily since $X_i = \tau - x_i$. For the extra clock x_0, we introduce variable X_0 equal to the global timestamp τ (since x_0 must remain equal to 0). A constraint graph defining a zone is a weighted graph whose nodes are $X = \{X_0, X_1, \ldots, X_n\}$. Constraints on clocks are represented by weights on edges in the graph: a constraint $X - Y \prec c$ is represented by an edge from X to Y weighted by $(\prec, c)$, with $\prec \in \{\leqslant, <\}$ and $c \in \mathbf{Q}$. Weights in the graph are thus pairs of the form $(\prec, c)$. Therefore, we can compute shortest weights between two vertices of a weighted graph. A cycle is said to be negative if it has weight at most $(<, 0)$, i.e. $(<, 0)$ or $(\prec, c)$ with $c < 0$.

Encoding Paths. Constraint graphs can also encode tuples of valuations seen along a path. To encode a k-step computation, we make $k+1$ copies of the nodes, that is, $X^i = \{X_0^i, X_1^i, \ldots, X_n^i\}$ for $i \in \{1, \ldots, k+1\}$. These copies are also called *layers*. Let us first consider an example on the path ρ consisting of the edge from ℓ_1 to ℓ_2, and the edge from ℓ_2 to ℓ_1, in the timed automaton of Fig. 1. The constraint graph G_ρ is depicted in Fig. 3: in our diagrams of constraint graphs, the absence of labels on an edge means $(\leqslant, 0)$, and we depict with an edge with arrows on both ends the presence of an edge in both directions. The graph has five columns, each containing copies of the variables for that step: they represent the valuations before the first edge, after the first time elapse, after the first reset, after the second time elapse and after the second reset. In general now, each elementary operation can be described by a constraint graph with two layers (X_i) (before) and (X_i') (after).

– The operation $\mathsf{Pretime}_{>t}$ is described by the constraint graph $G_{\mathsf{time}}^{>t}$ with edges $X_i \to X_0$, $X_i \leftrightarrow X_i'$ for $i > 0$, and $X_0 \xrightarrow{(<,-t)} X_0'$. Figure 3 contains two occurrences of $G_{\mathsf{time}}^{>0}$: we always represent with dashed arrows edges that are

labelled by $(<, c)$, and plain arrows edges that are labelled with $(\leqslant, c)$; the absence of an edge means that it is labelled with $(<, \infty)$.

– The operation $g \cap \mathsf{Unreset}_{\mathcal{Y}}(\cdot)$, to test a guard g and reset the clocks in $\mathcal{Y}$, is described by the constraint graph $G^{g,\mathcal{Y}}_{\mathsf{edge}}$ with edges $X_0 \leftrightarrow X'_0$ (meaning that the time does not elapse), $X_i \leftrightarrow X'_i$ for i such that clock $x_i \notin \mathcal{Y}$, and $X'_i \leftrightarrow X'_0$ for i such that clock $x_i \in \mathcal{Y}$, and for all clock constraint $x_i - x_j \prec c$ appearing in g, an edge from X_j to X_i labelled by $(\prec, c)$ (since it encodes the fact that $(\tau - X_i) - (\tau - X_j) = X_j - X_i \prec c$). In Fig. 3, we have first $G^{x_1 \leqslant 2, \{x_1\}}_{\mathsf{edge}}$, and then $G^{x_2 \geqslant 2, \{x_2\}}_{\mathsf{edge}}$.

Constraint graphs can be stacked one after the other to obtain the constraint graph of an edge e, and then of a path ρ, that we denote by G_ρ. In the resulting graph, there is one leftmost layer of vertices $(X^\ell_i)_i$ and one rightmost one $(X^r_i)_i$ representing the situation before and after the firing of the path ρ. Once this graph is constructed, the intermediary levels can be eliminated after replacing each edge between the nodes of $X^\ell \cup X^r$ by the shortest path in the graph. This phase is hereafter called *normalisation* of the constraint graph. The normalised version of the constraint graph of Fig. 3 is depicted on its right.

From Constraint Graphs to Reachability Relations. From a logical point of view, the elimination of intermediary layers reflects an elimination of quantifiers in a formula of the first-order theory of real numbers. At the end, we obtain a set of constraints of the form $X^k_i - X^{k'}_j \prec c$ with $k, k' \in \{\ell, r\}$. These constraints do not reflect uniquely the reachability relation $\mathsf{Reach}(\rho)$, in the sense that it is possible that $\mathsf{Reach}(\rho_1) = \mathsf{Reach}(\rho_2)$ but the normalised versions of G_{ρ_1} and G_{ρ_2} are different. For example, if we consider the path ρ^2 obtained by repeating the cycle ρ between ℓ_1 and ℓ_2, the reachability relation does not change $(\mathsf{Reach}(\rho^2) = \mathsf{Reach}(\rho))$, but the normalised constraint graph does $(G_{\rho^2} \neq G_{\rho^1})$: all labels $(\leqslant, 2)$ of the red dotted edges from the rightmost layer to the leftmost layer become $(\leqslant, 4)$, and the labels $(\leqslant, -2)$ of the dashed blue edges become $(\leqslant, -4)$.

We solve this issue by jumping back from variables X^k_i to the clock valuations. Indeed, in terms of clock valuations ν^ℓ and ν^r before and after the path, the constraint $X^k_i - X^{k'}_j \prec c$ (for $k, k' \in \{l, r\}$) rewrites as $(\tau^k - \nu^k(x_i)) - (\tau^{k'} - \nu^{k'}(x_j)) \prec c$, where τ^ℓ is the global timestamp before firing ρ and τ^r the one after. When $k = k'$, variables τ^ℓ and τ^r disappear, leaving a constraint of the form $\nu^k(x_j) - \nu^k(x_i) \prec c$. When $k \neq k'$, we can rewrite the constraint as $\tau^k - \tau^{k'} \prec \nu^k(x_i) - \nu^{k'}(x_j) + c$. We therefore obtain upper and lower bounds on the value of $\tau^r - \tau^\ell$, allowing us to eliminate $\tau^r - \tau^\ell$ considered as a single variable. We therefore obtain in fine a formula mixing constraints of the form

- $\nu^k(x_a) - \nu^k(x_b) \prec p$, with $k \in \{\ell, r\}$, $a \neq b$, and we define $\gamma^k_{a,b} = (\prec, p)$;
- $\nu^\ell(x_a) - \nu^\ell(x_b) + \nu^r(x_c) - \nu^r(x_d) \prec p$, with $a \neq b$ and $c \neq d$, and we define $\gamma_{a,b,c,d} = (\prec, p)$. This constraint can appear in two ways: either from $\nu^r(x_c) - \nu^\ell(x_b) + p_1 \prec_1 \tau^r - \tau^l \prec_2 \nu^l(x_a) - \nu^r(x_d) + p_2$ by eliminating $\tau^r - \tau^l$, or by adding the two constraints of the form $\nu^l(x_a) - \nu^l(x_b) \prec_1 p_1$ and

$\nu^r(x_c) - \nu^r(x_d) \prec_2 p_2$. Thus, $\gamma_{a,b,c,d}$ is obtained as the minimum of the two constraints obtained in this manner. In other terms, in the constraint graph, this constraint is the minimal weight between the sum of the weights of the edges (X^r_d, X^l_a) and (X^l_b, X^r_c), and the sum of the weights of the edges (X^l_b, X^l_a) and (X^r_d, X^r_c). For example, in the path in Fig. 3, we have $\gamma_{0,1,0,2} = (\leqslant, 0)$ since the two constraints are $(\leqslant, 0)$ and $(<, \infty)$, whereas $\gamma_{1,2,2,1} = (\leqslant, 0)$ because the two constraints are $(<, 2)$ and $(\leqslant, 0)$.

Let $\varphi(G)$ be the conjunction of such constraints obtained from a constraint graph G once normalised: this is a quantifier-free formula of the additive theory of reals. We obtain the following property whose proof mimics the one for proving the normalisation of DBMs (and can be derived from the developments of [8]).

Lemma 1. *Let ρ be a path in a timed automaton. If G_ρ contains a negative cycle, then $\mathsf{Reach}(\rho) = \emptyset$. Otherwise, $\mathsf{Reach}(\rho)$ is the set of pairs of valuations (ν^ℓ, ν^r) that satisfy the formula $\varphi(G_\rho)$.*

Checking Inclusion. For a path ρ, we regroup the pairs $(\gamma^l_{a,b})$, $(\gamma^r_{a,b})$ and $(\gamma_{a,b,c,d})$ above in a single vector Γ^ρ. We extend the comparison relation $<$ to these vectors by applying it componentwise. These vectors can be used to check equality or inclusion of reachability relations in time $O(|X|^4)$:

Theorem 1. *Let ρ and ρ' be paths in a timed automaton such that $\mathsf{Reach}(\rho)$ and $\mathsf{Reach}(\rho')$ are non empty. Then $\mathsf{Reach}(\rho) \subseteq \mathsf{Reach}(\rho')$ if and only if $\Gamma^\rho \leqslant \Gamma^{\rho'}$.*

Notice that we do not need to check equivalence or implication of formulas $\varphi(G_\rho)$ and $\varphi(G_{\rho'})$, but simply check syntactically constants appearing in these formulas. Moreover, these constants can be stored in usual DBMs on $2 \times |\mathcal{X}_0|$ clocks, allowing for reusability of classical DBM libraries. For the constraint graph in Fig. 3, we have seen that $G_{\rho^2} \neq G_{\rho^1}$, even if $\mathsf{Reach}(\rho^2) = \mathsf{Reach}(\rho)$. However, we can check that $\varphi(G_{\rho^2}) = \varphi(G_\rho)$ as expected.

Computation of Pre and Post. By Lemma 1 and the construction of constraint graphs, one can easily compute $\mathsf{Pre}_\rho(Z) = \{\nu \mid \exists \nu' \in Z\ ((\ell, \nu), (\ell', \nu')) \in \mathsf{Reach}(\rho)\}$ for a given path ρ and zone Z (see [8,16]). In fact, consider the normalised constraint graph G_ρ on nodes $X^\ell \cup X^r$. To compute $\mathsf{Pre}_\rho(Z)$, one just needs to add the constraints of Z on X^r. This is done by replacing each edge $X^r_i \xrightarrow{w} X^r_j$ by $X^r_i \xrightarrow{\min(Z_{j,i}, w)} X^r_j$ where $Z_{j,i} = (\prec, p)$ defines the constraint of Z on $x_j - x_i$. Then, the normalisation of the graph describes the reachability relation along path ρ ending in zone Z. Furthermore, projecting the constraints to X^ℓ yields $\mathsf{Pre}_\rho(Z)$: this can be obtained by gathering all constraints on pairs of nodes of X^ℓ. A reachability relation can thus be seen as a function assigning to each zone Z its image by ρ. One can symmetrically compute the successor $\mathsf{Post}_\rho(Z) = \{\nu' \mid \exists \nu \in Z\ ((\ell, \nu), (\ell', \nu')) \in \mathsf{Reach}(\rho)\}$ by constraining the nodes X^ℓ and projecting to X^r.

4 Robust Iterability of a Lasso

In this section, we study the perturbation game $\mathcal{G}_\delta(\mathcal{A})$ between the two players (controller and environment), as defined in Sect. 2, when the timed automaton $\mathcal{A}$ is restricted to a fixed *lasso* $\rho_1\rho_2$, i.e. ρ_1 is a path from ℓ_0 to some accepting location ℓ_t, and ρ_2 a cyclic path around ℓ_t. This implies that the controller does not have the choice of the transitions, but only of the delays. We will consider different settings, in which δ is fixed or not.

Controllable Predecessors and their Greatest Fixpoints. Consider an edge $e = (\ell, g, R, \ell')$. For any set $Z \subseteq \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$, we define the *controllable predecessors of* Z as follows: $\mathsf{CPre}^\delta_e(Z) = \mathsf{Pretime}_{>\delta}(\mathsf{shrink}_{[-\delta,\delta]}(g \cap \mathsf{Unreset}_R(Z)))$. Intuitively, $\mathsf{CPre}^\delta_e(Z)$ is the set of valuations from which the controller can ensure reaching Z in one step, following the edge e, no matter of the perturbations of amplitude at most δ of the environment. In fact, it can delay in $\mathsf{shrink}_{[-\delta,\delta]}(g \cap \mathsf{Unreset}_R(Z))$ with a delay of at least δ, where under any perturbation in $[-\delta, \delta]$, the valuation satisfies the guard, and it ends, after reset, in Z. Results of [24] show that this operator can be computed in cubic time with respect to the number of clocks. We extend this operator to a path ρ by composition, denoted it by $\mathsf{CPre}^\delta_\rho$. Note that $\mathsf{CPre}^0_\rho = \mathsf{Pre}_\rho$ is the usual predecessor operator without perturbation.

This operator is monotone, hence its greatest fixpoint $\nu X\, \mathsf{CPre}^\delta_\rho(X)$ is well-defined, equal to $\bigcap_{i\geqslant 0} \mathsf{CPre}^\delta_{\rho^i}(\top)$: it corresponds to the valuations from which the controller can guarantee to loop forever along the path ρ. By definition of the game $\mathcal{G}_\delta(\mathcal{A})$ where $\mathcal{A}$ is restricted to the lasso $\rho_1\rho_2$, the controller wins the game if and only if $\mathbf{0} \in \mathsf{CPre}^\delta_{\rho_1}(\nu X\, \mathsf{CPre}^\delta_{\rho_2}(X))$. As a consequence, our problem reduces to the computation of this greatest fixpoint.

Branching Constraint Graphs. We consider first a fixed (rational) value of the parameter δ, and are interested in the computation of the greatest fixpoint $\nu X\, \mathsf{CPre}^\delta_{\rho_2}(X)$. In [16], constraints graphs were used to provide a termination criterion allowing to compute the greatest fixpoint of the classical predecessor operator CPre^0_ρ. We generalize this approach to deal with the operator $\mathsf{CPre}^\delta_\rho$ and to this end, we need to generalize constraint graphs so as to encode it. Unfortunately, the operator $\mathsf{shrink}_{[-\delta,\delta]}$ cannot be encoded in a constraint graph. Intuitively, this comes from the fact that a constraint graph represents a relation between valuations, while there is no such relation associated with the $\mathsf{CPre}^\delta_\rho$ operator. Instead, we introduce *branching constraint graphs*, that will faithfully represent the $\mathsf{CPre}^\delta_\rho$ operator: unlike constraint graphs introduced so far that have a left layer and a right layer of variables, a branching constraint graph has still a single left layer but several right layers.

We first define a branching constraint graph $G^\delta_{\mathsf{shrink}}$ associated with the operator $\mathsf{shrink}_{[-\delta,\delta]}$ as follows. Its set of vertices is composed of three copies of the $\{X_0, X_1, \ldots, X_n\}$, denoted by primed, unprimed and doubly primed versions. Edges are defined so as to encode the following constraints : $X'_i = X_i$ and $X''_i = X_i$ for every $i \neq 0$, and $X'_0 = X_0 + \delta$ and $X''_0 = X_0 - \delta$. An instance of this graph can be found in several occurrences in Fig. 2.

Proposition 1. *Let Z be a zone and $G^{\delta}_{\mathsf{shrink}}(Z)$ be the graph obtained from $G^{\delta}_{\mathsf{shrink}}$ by adding on primed and doubly primed vertices the constraints defining Z (as for $\mathsf{Pre}_{\rho}(Z)$ in the end of Sect. 3). Then the constraint on unprimed vertices obtained from the shortest paths in $G^{\delta}_{\mathsf{shrink}}(Z)$ is equivalent to $\mathsf{shrink}_{[-\delta,\delta]}(Z)$.*

Proof. Given a zone Z and a real number d, we define $Z + d = \{\nu + d \mid \nu \in Z\}$. One easily observes that $\mathsf{shrink}_{[-\delta,\delta]}(Z) = (Z + \delta) \cap (Z - \delta)$. The result follows from the observation that taking two distinct copies of vertices, and considering shortest paths allows one to encode the intersection. □

Then, for all edges $e = (\ell, g, R, \ell')$, we define the branching constraint graph G^{δ}_{e} as the graph obtained by stacking (in this order) the branching constraint graph $G^{>\delta}_{\mathsf{time}}$, $G^{\delta}_{\mathsf{shrink}}$ and $G^{g,\mathcal{Y}}_{\mathsf{edge}}$. Note that two copies of the graph $G^{g,\mathcal{Y}}_{\mathsf{edge}}$ are needed, to be connected to the two sets of vertices that are on the right of the graph $G^{\delta}_{\mathsf{shrink}}$. This definition is extended in the expected way to a finite path ρ, yielding the graph G^{δ}_{ρ}. In this graph, there is a single set of vertices on the left, and $2^{|\rho|}$ sets of vertices on the right. As a direct consequence of the previous results on the constraint graphs for time elapse, shrinking and guard/reset, one obtains:

Proposition 2. *Let Z be a zone and ρ be a path. We let $G^{\delta}_{\rho}(Z)$ be the graph obtained from G^{δ}_{ρ} by adding on every set of right vertices the constraints defining Z. Then the constraint on the left layer of vertices obtained from the shortest paths in $G^{\delta}_{\rho}(Z)$ is equivalent to $\mathsf{CPre}^{\delta}_{\rho}(Z)$.*

An example of the graph $G^{\delta}_{\rho}(Z)$ for $\rho = e_1 e_2$, edges considered in Fig. 3, is depicted in Fig. 2 (on the left).

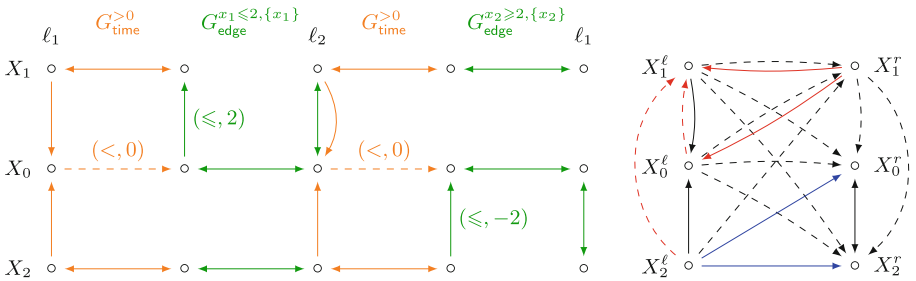


Fig. 2. On the left, the branching constraint graph $G^{\delta}_{e_1 e_2}$ encoding the operator $\mathsf{CPre}^{\delta}_{e_1 e_2}$, where e_1 and e_2 refer to edges considered in Fig. 3. Dashed edges have weight $(<, .)$, plain edges have weight $(\leqslant, .)$. Black edges (resp. orange edges, pink edges, red edges, blue edges) are labelled by $(., 0)$ (resp. $(., -\delta)$, $(., \delta)$, $(., 2)$,$(., -2)$). On the right, a decomposition of a path in a branching constraint graph G^{δ}_{ρ}. (Color figure online)

We are now ready to prove the following result, generalisation of [16, Lemma 2], that will allow us to compute the greatest fixpoint of the operator $\mathsf{CPre}^{\delta}_{\rho}$:

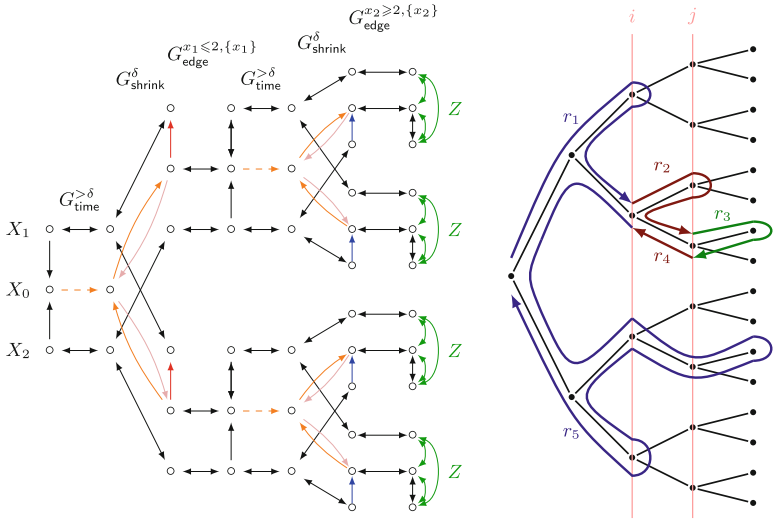


Fig. 3. On the left, the constraint graph of the path $\ell_1 \xrightarrow{x_1\leqslant 2, x_1:=0} \ell_2 \xrightarrow{x_2\geqslant 2, x_2:=0} \ell_1$. On the right, its normalised version: dashed edges have weight $(<, .)$, plain edges have weight $(\leqslant, .)$, black edges have weight $(., 0)$, red edges have weight $(., 2)$ and blue edges have weight $(., -2)$.

Proposition 3. *Let ρ be a path and δ be a non-negative rational number. We let $N = |\mathcal{X}_0|^2$. If $\mathsf{CPre}^{\delta}_{\rho^{2N+1}}(\top) \subsetneq \mathsf{CPre}^{\delta}_{\rho^{2N}}(\top)$, then $\nu X\, \mathsf{CPre}^{\delta}_{\rho}(X) = \emptyset$.*

Proof. Assume $\mathsf{CPre}^{\delta}_{\rho^{2N+1}}(\top) \subsetneq \mathsf{CPre}^{\delta}_{\rho^{2N}}(\top)$ and consider the zones $\mathsf{CPre}^{\delta}_{\rho^{N+1}}(\top)$ (represented by the DBM M_1) and $\mathsf{CPre}^{\delta}_{\rho^{N}}(\top)$ (represented by the DBM M_2). We have $M_1 \subsetneq M_2$, as otherwise the fixpoint would have already been reached after N steps. By Proposition 2, the zone corresponding to M_1 is associated with shortest paths between vertices on the left in the graph $G^{\delta}_{\rho^{N+1}}$. In the sequel, given a path r in this graph, $w(r)$ denotes its weight. We distinguish two cases:

Case 1: $M_1 \subsetneq M_2$ because of the rational coefficients. Then, there exists an entry $(x, y) \in \mathcal{X}_0^2$ such that $\mathsf{M}_1[x, y] < \mathsf{M}_2[x, y]$. The value $M_1[x, y]$ is thus associated with a shortest path between vertices X and Y in $G^{\delta}_{\rho^{N+1}}$. We fix a shortest path of minimal length, and denote it by r. As the entry is strictly smaller than in M_2, this shortest path should reach the last copy of the graph G^{δ}_{ρ}. This path can be interpreted as a traversal of the binary tree of depth $|\mathcal{X}_0|^2 + 1$, reaching at least one leaf. We can prove that this entails that there exists a pair of clocks $(u, v) \in \mathcal{X}_0^2$ appearing at two levels $i < j$ of this tree, and a decomposition $r = r_1 r_2 r_3 r_4 r_5$ of the path, such that $w(r_2) + w(r_4) = (\prec, d)$ with $d < 0$ (Property (†)). In addition, in this decomposition, r_3 is included in subgraphs of levels $k \geq j$, and the pair of paths (r_2, r_4) is called a *return path*, following the terminology of [16]. This decomposition is depicted in Fig. 2 (on the right). Intuitively, the property (†) follows from the fact that as r_3 is

included in subgraphs of levels $k \geq j$, and because the final zone (on the right) is the zone $\top$ which adds no edges, the concatenation $r' = r_1 r_3 r_5$ is also a valid path from X to Y in $G^{\delta}_{\rho^{N+1}}$, and is shorter than r. We conclude using the fact that r has been chosen as a shortest path of minimal weight.

Property (†) allows us to prove that the greatest fixpoint is empty. Indeed, by considering iterations of ρ, one can repeat the return path associated with (r_2, r_4) and obtain paths from X to Y whose weights diverge towards $-\infty$.

Case 2: $M_1 \subsetneq M_2$ because of the ordering coefficients. We claim that this case cannot occur. Indeed, one can show that the constants will not evolve anymore after the Nth iteration of the fixpoint: the coefficients can only decrease by changing from a non-strict inequality $(\leq, c)$ to a strict one $(<, c)$. This propagation of strict inequalities is performed in at most $|\mathcal{X}_0|^2$ additional steps, thus we have $\mathsf{CPre}^{\delta}_{\rho^{2N+1}}(\top) = \mathsf{CPre}^{\delta}_{\rho^{2N}}(\top)$, yielding a contradiction. □

Compared to the result of [16], the number of iterations needed before convergence grows from $|\mathcal{X}_0|^2$ to $2|\mathcal{X}_0|^2$: this is due to the presence of strict and non-strict inequalities, not considered in [16]. With the help of branching constraint graphs, we have thus shown that the greatest fixpoint can be computed in finite time: this can then be done directly with computations on zones (and not on branching constraint graphs).

Proposition 4. *Given a path ρ and a rational number δ, the greatest fixpoint $\nu X\, \mathsf{CPre}^{\delta}_{\rho}(X)$ can be computed in time polynomial in $|\mathcal{X}|$ and $|\rho|$. As a consequence, one can decide whether the controller has a strategy along a lasso $\rho_1\rho_2$ in $\mathcal{G}_{\delta}(\mathcal{A})$ in time polynomial in $|\mathcal{X}|$ and $|\rho_1\rho_2|$.*

Solving the Robust Controller Synthesis Problem for a Lasso. We have shown how to decide whether the controller has a winning strategy for a fixed rational value of δ. We now aim at deciding whether there exists a positive value of δ for which the controller wins the game $\mathcal{G}_{\delta}(\mathcal{A})$ (where $\mathcal{A}$ is restricted to a lasso $\rho_1\rho_2$). To this end, we will use a parametrised extension of DBMs, namely *shrunk DBMs*, that were introduced in [24] in order to study the parametrised state space of timed automata. Intuitively, our goal is to express *shrinkings* of guards, e.g. sets of states satisfying constraints of the form $g = 1 + \delta < x < 2 - \delta \wedge 2\delta < y$, where δ is a parameter to be chosen. Formally, a shrunk DBM is a pair (M, P), where M is a DBM, and P is a nonnegative integer matrix called a *shrinking matrix*. This pair represents the set of valuations defined by the DBM $M - \delta P$, for any given $\delta > 0$. Considering the example g, M is the guard g obtained by setting $\delta = 0$, and P is made of the integer multipliers of δ. We adopt the following notation: when we write a statement involving a shrunk DBM (M, P), we mean that for some $\delta_0 > 0$, the statement holds for $M - \delta P$ for all $\delta \in (0, \delta_0]$. For instance, $(M, P) = \mathsf{Pretime}_{>\delta}((N, Q))$ means that $M - \delta P = \mathsf{Pretime}_{>\delta}(N - \delta Q)$ for all small enough $\delta > 0$. Shrunk DBMs are closed under standard operations on zones, and as a consequence, the CPre operator can be computed on shrunk DBMs:

Lemma 2. ([25]) *Let $e = (\ell, g, R, \ell')$ be an edge and (M, P) be a shrunk DBM. Then, there exists a shrunk DBM (N, Q), that we can compute in polynomial time, such that $(N, Q) = \mathsf{CPre}_e^{\delta}((M, P))$.*

Proposition 5. *Given a path ρ, one can compute a shrunk DBM (M, P) equal to the greatest fixpoint of the operator $\mathsf{CPre}_\rho^{\delta}$. As a consequence, one can solve the parametrised robust controller synthesis problem for a given lasso in time complexity polynomial in the number of clocks and in the length of the lasso.*

Proof. The bound $2|\mathcal{X}_0|^2$ identified previously does not depend on the value of δ. Hence the algorithm for computing a shrunk DBM representing the greatest fixpoint proceeds as follows. It computes symbolically, using shrunk DBMs, the $2|\mathcal{X}_0|^2$-th and $2|\mathcal{X}_0|^2 + 1$-th iterations of the operator $\mathsf{CPre}_\rho^{\delta}$, from the zone $\top$. By monotonicity, the $2|\mathcal{X}_0|^2 + 1$-th iteration is included in the $2|\mathcal{X}_0|^2$-th. If the two shrunk DBMs are equal, then they are also equal to the greatest fixpoint. Otherwise, the greatest fixpoint is empty. To decide the robust controller synthesis problem for a given lasso, one first computes a shrunk DBM representing the greatest fixpoint associated with ρ_2 and, if not empty, one computes a new shrunk DBM by applying to it the operator $\mathsf{CPre}_{\rho_1}^{\delta}$. Then, one checks whether the valuation $\mathbf{0}$ belongs to the resulting shrunk DBM. □

Computing the Largest Admissible Perturbation. We say that a perturbation δ is *admissible* if the controller wins the game $\mathcal{G}_\delta(\mathcal{A})$. The parametrised robust controller synthesis problem, solved before just for a lasso, aims at deciding whether *there exists* a positive admissible perturbation. A more ambitious problem consists in determining the *largest admissible* perturbation.

The previous algorithm performs a bounded $(2|\mathcal{X}_0|^2)$ number of computations of the $\mathsf{CPre}_\rho^{\delta}$ operator. Instead of focusing on arbitrarily small values using shrunk DBMs as we did previously, we must perform a computation for all values of δ. To do so, we consider an extension of the (shrunk) DBMs in which each entry of the matrix (which thus represents a clock constraint) is a piecewise affine function of δ. One can observe that all the operations involved in the computation of the $\mathsf{CPre}_\rho^{\delta}$ operator can be performed symbolically w.r.t. δ using piecewise affine functions. As a consequence, we obtain the following new result:

Proposition 6. *We can compute the largest admissible perturbation of a lasso.*

Proof. Let $\rho_1\rho_2$ be a lasso. One first computes a symbolic representation, valid for all values of δ, of the greatest fixpoint of $\mathsf{CPre}_{\rho_2}^{\delta}$. To do so, one computes the $2|\mathcal{X}_0|^2$-th and $2|\mathcal{X}_0|^2+1$-th iterations of this operator, from the zone $\top$. We denote them by M_1 and M_2 respectively. By monotonicity, the inclusion $M_1(\delta) \subseteq M_2(\delta)$ holds for every $\delta \geq 0$. In addition, both M_1 and M_2 are decreasing w.r.t. δ, thus one can identify the value $\delta_0 = \inf\{\delta \geq 0 \mid M_1(\delta) \subsetneq M_2(\delta)\}$. Then, the greatest fixpoint is equal to M_1 for $\delta < \delta_0$, and to the emptyset for δ at least δ_0. As a second step, one applies the operator CPre_{ρ_1} to the greatest fixpoint. We denote the result by M. To conclude, one can then compute and return the value $\sup\{\delta \in [0, \delta_0[\mid \mathbf{0} \in M(\delta)\}$ of maximal perturbation. □

5 Synthesis of Robust Controllers

We are now ready to solve the parametrised robust controller synthesis problem, that is to find, if it exists, a lasso $\rho_1\rho_2$ and a perturbation δ such that the controller wins the game $\mathcal{G}_\delta(\mathcal{A})$ when following the lasso $\rho_1\rho_2$ as a strategy. As for the symbolic checking of emptiness of a Büchi timed language [17], we will use a double forward analysis to exhaust all possible lassos, each being tested for robustness by the techniques studied in previous section: a first forward analysis will search for ρ_1, a path from the initial location to an accepting location, and a second forward analysis from each accepting location ℓ to find the cycle ρ_2 around ℓ. Forward analysis means that we compute the successor zone $\mathsf{Post}_\rho(Z)$ when following path ρ from zone Z.

Abstractions of Lassos. Before studying in more details the two independent forward analyses, we first study what information we must keep about ρ_1 and ρ_2 in order to still being able to test the robustness of the lasso $\rho_1\rho_2$. A classical problem for robustness is the firing of a *punctual transition*, i.e. a transition where controller has a single choice of time delay: clearly such a firing will be robust for no possible choice of parameter δ. Therefore, we must at least forbid such punctual transitions in our forward analyses. We thus introduce a non-punctual successor operator $\mathsf{Post}^{\mathrm{np}}_\rho$. It consists of the standard successor operator Post_ρ in the timed automaton $\mathcal{A}^{\mathrm{np}}$ obtained from $\mathcal{A}$ by making strict every constraint appearing in the guards ($1 \leq x \leq 2$ becomes $1 < x < 2$). The crucial point is that if a positive delay d can be taken by the controller while satisfying a set of strict constraints, then other delays are also possible, close enough to d. By analogy, a region is said to be *non-punctual* if it contains two valuations separated by a positive time delay. In particular, if such a region satisfies a constraint in $\mathcal{A}$ it also satisfies the corresponding strict constraint in $\mathcal{A}^{\mathrm{np}}$. Therefore, controller wins $\mathcal{G}_\delta(\mathcal{A})$ for some $\delta > 0$ if and only if he wins $\mathcal{G}_\delta(\mathcal{A}^{\mathrm{np}})$ for some $\delta > 0$.

The link between non-punctuality and robustness is as follows:

Theorem 2. *Let $\rho_1\rho_2$ be a lasso of the timed automaton. We have*

$$\exists \delta > 0 \;\; \mathbf{0} \in \mathsf{CPre}^\delta_{\rho_1}(\nu X\, \mathsf{CPre}^\delta_{\rho_2}(X)) \iff \mathsf{Post}^{\mathrm{np}}_{\rho_1}(\mathbf{0}) \cap (\textstyle\bigcup_{\delta>0} \nu X\, \mathsf{CPre}^\delta_{\rho_2}(X)) \neq \emptyset$$

Proof. The proof of this theorem relies on three main ingredients:

1. the timed automaton $\mathcal{A}^{\mathrm{np}}$ allows one to compute $\bigcup_{\delta>0} \mathsf{CPre}^\delta_e(Z')$ by classical predecessor operator: $\mathsf{Pre}^{\mathrm{np}}_e(Z') = \bigcup_{\delta>0} \mathsf{CPre}^\delta_e(Z')$;
2. for all edges e, and zones Z and Z', $Z \cap \mathsf{Pre}^{\mathrm{np}}_e(Z') \neq \emptyset$ if and only if $\mathsf{Post}^{\mathrm{np}}_e(Z) \cap Z' \neq \emptyset$: this duality property on predecessor and successor relations always holds, in particular in $\mathcal{A}^{\mathrm{np}}$. These two ingredients already imply that the theorem holds for a path reduced to a single edge e;
3. we then prove the theorem by induction on length of the path using that $\bigcup_{\delta>0} \mathsf{CPre}^\delta_{\rho_1\rho_2}(Z) = \bigcup_{\delta>0} \mathsf{CPre}^\delta_{\rho_1}\big(\bigcup_{\delta'>0} \mathsf{CPre}^{\delta'}_{\rho_2}(Z)\big)$, due to the monotonicity of the $\mathsf{CPre}^\delta_{\rho_1}$ operator. □

Therefore, in order to test the robustness of the lasso $\rho_1\rho_2$, it is enough to only keep in memory the sets $\mathsf{Post}^{\mathrm{np}}_{\rho_1}(\mathbf{0})$ and $\bigcup_{\delta>0} \nu X\, \mathsf{CPre}^{\delta}_{\rho_2}(X)$.

Non-punctual Forward Analysis. As a consequence of the previous theorem, we can use a classical forward analysis of the timed automaton $\mathcal{A}^{\mathrm{np}}$ to look for the prefix ρ_1 of the lasso $\rho_1\rho_2$. A classical inclusion check on zones allows to stop the exploration, this criterion being complete thanks to Theorem 2. It is worth reminding that we consider only bounded clocks, hence the number of reachable zones is finite, ensuring termination.

Robust Cycle Search. We now perform a second forward analysis, from each possible final location, to find a robust cycle around it. To this end, for each cycle ρ_2, we must compute the zone $\bigcup_{\delta>0} \nu X\, \mathsf{CPre}^{\delta}_{\rho_2}(X)$. This computation is obtained by arguments developed in Sect. 4 (Proposition 4). To enumerate cycles ρ_2, we can again use a classical forward exploration, starting from the universal zone $\top$. Using zone inclusion to stop the exploration is not complete: considering a path ρ_2' reaching a zone Z_2' included in the zone Z_2 reachable using some ρ_2, ρ_2' could be robustly iterable while ρ_2 is not. In order to ensure termination of our analysis, we instead use reachability relations inclusion checks. These tests are performed using the technique developed in Sect. 3, based on constraint graphs (Theorem 1). The correction of this inclusion check is stated in the following lemma, where $\mathsf{Reach}^{\mathrm{np}}_{\rho}$ denotes the reachability relation associated with ρ in the automaton $\mathcal{A}^{\mathrm{np}}$. This result is derived from the analysis based on regions in [25]. Indeed, we can prove that the non-punctual reachability relation we consider captures the existence of non-punctual aperiodic paths in the region automaton, as considered in [25].

Lemma 3. *Let ρ_1 a path from ℓ_0 to some target location ℓ_t. Let ρ_2, ρ_2' be two paths from ℓ_t to some location ℓ, such that $\mathsf{Reach}^{\mathrm{np}}_{\rho_2} \subseteq \mathsf{Reach}^{\mathrm{np}}_{\rho_2'}$. For all paths ρ_3 from ℓ to ℓ_t, $\mathsf{Post}^{\mathrm{np}}_{\rho_1}(\mathbf{0}) \cap (\bigcup_{\delta>0} \nu X\, \mathsf{CPre}^{\delta}_{\rho_2\rho_3}(X)) \neq \emptyset$ implies $\mathsf{Post}^{\mathrm{np}}_{\rho_1}(\mathbf{0}) \cap (\bigcup_{\delta>0} \nu X\, \mathsf{CPre}^{\delta}_{\rho_2'\rho_3}(X)) \neq \emptyset$.*

6 Case Study

We implemented our algorithm in C++. To illustrate our approach, we present a case study on the regulation of train networks. Urban train networks in big cities are often particularly busy during rush hours: trains run in high frequency so even small delays due to incidents or passenger misbehavior can perturb the traffic and end up causing large delays. Train companies thus apply regulation techniques: they slow down or accelerate trains, and modify waiting times in order to make sure that the traffic is fluid along the network. Computing robust schedules with provable guarantees is a difficult problem (see e.g. [9]).

We study here a simplified model of a train network and aim at automatically synthesizing a controller that regulates the network despite perturbations, in order to ensure performance measures on total travel time for each train. Consider a circular train network with m stations $s_0, \ldots, s_{m-1}$ and n trains. We

require that all trains are at distinct stations at all times. There is an interval of delays $[\ell_i, u_i]$ attached to each station which bounds the travel time from s_i to $s_{i+1 \bmod m}$. Here the lower bound comes from physical limits (maximal allowed speed, and travel distance) while the upper bound comes from operator specification (e.g. it is not desirable for a train to remain at station for more than 3 min). The objective of each train i is to cycle on the network while completing each tour within a given time interval $[t_1^i, t_2^i]$.

All timing requirements are naturally encoded with clocks. Given a model, we solve the robust controller synthesis problem in order to find a controller choosing travel times for all trains ensuring a Büchi condition (visiting s_1 infinitely often). Given the fact that trains cannot be at the same station at any given time, it suffices to state the Büchi condition only for one train, since its satisfaction of the condition necessarily implies that of all other trains.

Let us present two representative instances and then comment the performance of the algorithm on a set of instances. Consider a network with two trains and m stations, with $[\ell_i, u_i] = [200, 400]$ for each station i, and the objective of both trains is the interval $[250 \cdot m, 350 \cdot m]$, that is, an average travel time between stations that lies in $[250, 350]$. The algorithm finds an accepting lasso: intuitively, by choosing δ small enough so that $m\delta < 50$, perturbations do not accumulate too much and the controller can always choose delays for both trains and satisfy the constraints. This case corresponds to scenario A in Fig. 4. Consider now the same network but with two different objectives: $[0, 300 \cdot m]$ and $[300 \cdot m, \infty)$. Thus, one train needs to complete each cycle in at most $300 \cdot m$ time units, while the other one in at least $300 \cdot m$ time units. A classical Büchi emptiness check reveals the existence of an accepting lasso: it suffices to move each train in exactly 300 time units between each station. This controller can even recover from perturbations for a bounded number of cycles: for instance, if a train arrives late at a station, the next travel time can be chosen smaller than 300. However, such corrections will cause the distance between the two trains to decrease and if such perturbations happen regularly, the system will eventually enter a deadlock. Our algorithm detects that there is no robust controller for the Büchi objective. This corresponds to the scenario B in Fig. 4.

Scenario	m	n	#Clocks	robust?	time
A	6	2	4	yes	4s
B	6	2	4	no	2s
C	6	3	5	no	263s
D	6	3	4	yes	125s
E	6	4	2	yes	53s
F	6	4	2	yes	424s
G	6	4	8		TO
H	6	4	8		TO
I	20	2	2	yes	76s
J	20	2	2	yes	55s
K	30	2	2	yes	579s

Fig. 4. Summary of experiments with different sizes. In each scenario, we assign a different objective to a subset of trains. The answer is *yes* if a robust controller was found, *no* if none exists. TO stands for a time-out of 30 min.

Figure 4 summarizes the outcome of our prototype implementation on other scenarios. The tool was run on a 3.2 Ghz Intel i7 processor running Linux, with

a 30 min time out and 2 GB of memory. The performance is sensitive to the number of clocks: on scenarios with 8 clocks the algorithm ran out of time.

7 Conclusion

Our case study illustrates the application of robust controller synthesis in small or moderate size problems. Our prototype relies on the DBM libraries that we use with twice as many clocks to store the constraints of the normalised constraint graphs. In order to scale to larger models, we plan to study extrapolation operators and their integration in the computation of reachability relations, which seems to be a challenging task. Different strategies can also be adopted for the double forward analysis, switching between the two modes using heuristics, a parallel implementation, etc.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
2. Bacci, G., Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Reynier, P.-A.: Optimal and robust controller synthesis. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) FM 2018. LNCS, vol. 10951, pp. 203–221. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95582-7_12
3. Bengtsson, J., Yi, W.: Timed automata: semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27755-2_3
4. Berthomieu, B., Menasche, M.: An enumerative approach for analyzing time Petri nets. In: Mason, R.E.A. (ed.) Information Processing 83 - Proceedings of the 9th IFIP World Computer Congress (WCC'83), pp. 41–46. North-Holland/IFIP, September 1983
5. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005). https://doi.org/10.1007/11539452_9
6. Cassez, F., Henzinger, T.A., Raskin, J.-F.: A comparison of control problems for timed and hybrid systems. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 134–148. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45873-5_13
7. Chatterjee, K., Henzinger, T.A., Prabhu, V.S.: Timed parity games: complexity and robustness. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 124–140. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85778-5_10
8. Comon, H., Jurski, Y.: Timed automata and the theory of real numbers. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 242–257. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48320-9_18
9. D'Ariano, A., Pranzo, M., Hansen, I.A.: Conflict resolution and train speed coordination for solving real-time timetable perturbations. IEEE Trans. Intell. Trans. Syst. **8**(2), 208–222 (2007)

10. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52148-8_17
11. Henzinger, T.A., Otop, J., Samanta, R.: Lipschitz robustness of timed I/O systems. In: Jobstmann, B., Leino, K.R.M. (eds.) VMCAI 2016. LNCS, vol. 9583, pp. 250–267. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49122-5_12
12. Henzinger, T.A., Sifakis, J.: The embedded systems design challenge. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 1–15. Springer, Heidelberg (2006). https://doi.org/10.1007/11813040_1
13. Herbreteau, F., Srivathsan, B.: Efficient on-the-fly emptiness check for timed büchi automata. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 218–232. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15643-4_17
14. Herbreteau, F., Srivathsan, B., Tran, T.-T., Walukiewicz, I.: Why liveness for timed automata is hard, and what we can do about it. In: FSTTCS 2016, LIPIcs, vol. 65, pp. 48:1–48:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
15. Herbreteau, F., Srivathsan, B., Walukiewicz, I.: Efficient emptiness check for timed büchi automata. Formal Methods Syst. Des. **40**(2), 122–146 (2012)
16. Jaubert, R., Reynier, P.-A.: Quantitative robustness analysis of flat timed automata. In: Hofmann, M. (ed.) FoSSaCS 2011. LNCS, vol. 6604, pp. 229–244. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19805-2_16
17. Laarman, A., Olesen, M.C., Dalsgaard, A.E., Larsen, K.G., van de Pol, J.: Multi-core emptiness checking of timed büchi automata using inclusion abstraction. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 968–983. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_69
18. Larsen, K.G., Legay, A., Traonouez, L.-M., Wasowski, A.: Robust synthesis for real-time systems. Theor. Comput. Sci. **515**, 96–122 (2014)
19. Li, G.: Checking timed büchi automata emptiness using LU-abstractions. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 228–242. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04368-0_18
20. Prabhakar, P., Soto, M.G.: Formal synthesis of stabilizing controllers for switched systems. In: Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017, New York, NY, USA, pp. 111–120. ACM (2017)
21. Prabhakar, P., Soto, M.G.: Counterexample guided abstraction refinement for stability analysis. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 495–512. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_27
22. Quaas, K., Shirmohammadi, M., Worrell, J.: Revisiting reachability in timed automata. In: LICS 2017. IEEE (2017)
23. Roohi, N., Prabhakar, P., Viswanathan, M.: Robust model checking of timed automata under clock drifts. In: Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017, New York, NY, USA, pp. 153–162. ACM (2017)
24. Sankur, O., Bouyer, P., Markey, N.: Shrinking timed automata. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011), LIPIcs, vol. 13, pp. 90–102. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2011)
25. Sankur, O., Bouyer, P., Markey, N., Reynier, P.-A.: Robust controller synthesis in timed automata. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013. LNCS, vol. 8052, pp. 546–560. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40184-8_38

26. Tran, T.-T.: Verification of timed automata : reachability, liveness and modelling. (Vérification d'automates temporisés : sûreté, vivacité et modélisation). Ph.D. thesis, University of Bordeaux, France (2016)
27. Tripakis, S.: Checking timed büchi automata emptiness on simulation graphs. ACM Trans. Comput. Log. **10**(3), 15:1–15:19 (2009)
28. Tripakis, S., Yovine, S., Bouajjani, A.: Checking timed büchi automata emptiness efficiently. Formal Methods Syst. Des. **26**(3), 267–292 (2005)

Flexible Computational Pipelines for Robust Abstraction-Based Control Synthesis

Eric S. Kim(✉), Murat Arcak, and Sanjit A. Seshia

UC Berkeley, Berkeley, CA, USA
{eskim,arcak,sseshia}@eecs.berkeley.edu

Abstract. Successfully synthesizing controllers for complex dynamical systems and specifications often requires leveraging domain knowledge as well as making difficult computational or mathematical tradeoffs. This paper presents a flexible and extensible framework for constructing robust control synthesis algorithms and applies this to the traditional abstraction-based control synthesis pipeline. It is grounded in the theory of relational interfaces and provides a principled methodology to seamlessly combine different techniques (such as dynamic precision grids, refining abstractions while synthesizing, or decomposed control predecessors) or create custom procedures to exploit an application's intrinsic structural properties. A Dubins vehicle is used as a motivating example to showcase memory and runtime improvements.

Keywords: Control synthesis · Finite abstraction · Relational interface

1 Introduction

A control synthesizer's high level goal is to automatically construct control software that enables a closed loop system to satisfy a desired specification. A vast and rich literature contains results that mathematically characterize solutions to different classes of problems and specifications, such as the Hamilton-Jacobi-Isaacs PDE for differential games [3], Lyapunov theory for stabilization [8], and fixed-points for temporal logic specifications [11,17]. While many control synthesis problems have elegant mathematical solutions, there is often a gap between a solution's theoretical characterization and the algorithms used to compute it. What data structures are used to represent the dynamics and constraints? What operations should those data structures support? How should the control synthesis algorithm be structured? Implementing solutions to the questions above can require substantial time. This problem is especially critical for computationally challenging problems, where it is often necessary to let the user *rapidly* identify and exploit structure through analysis or experimentation.

The authors were funded in part by AFOSR FA9550-18-1-0253, DARPA Assured Autonomy project, iCyPhy, Berkeley Deep Drive, and NSF grant CNS-1739816.

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 591–608, 2019.
https://doi.org/10.1007/978-3-030-25540-4_34

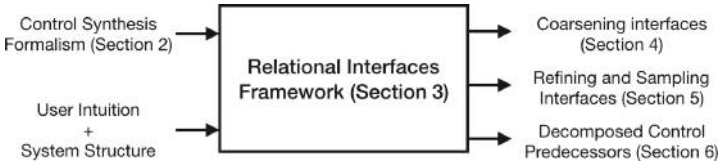


Fig. 1. By expressing many different techniques within a common framework, users are able to rapidly develop methods to exploit system structure in controller synthesis.

1.1 Bottlenecks in Abstraction-Based Control Synthesis

This paper's goal is to enable a framework to develop extensible tools for robust controller synthesis. It was inspired in part by computational bottlenecks encountered in control synthesizers that construct finite abstractions of continuous systems, which we use as a target use case. A traditional abstraction-based control synthesis pipeline consists of three distinct stages:

1. Abstracting the continuous state system into a finite automaton whose underlying transitions faithfully mimic the original dynamics [21,23].
2. Synthesizing a discrete controller by leveraging data structures and symbolic reasoning algorithms to mitigate combinatorial state explosion.
3. Refining the discrete controller into a continuous one. Feasibility of this step is ensured through the abstraction step.

This pipeline appears in tools PESSOA [12] and SCOTS [19], which can exhibit acute computational bottlenecks for high dimensional and nonlinear system dynamics. A common method to mitigate these bottlenecks is to exploit a specific dynamical system's topological and algebraic properties. In MASCOT [7] and CoSyMA [14], multi-scale grids and hierarchical models capture notions of state-space locality. One could incrementally construct an abstraction of the system dynamics while performing the control synthesis step [10,15] as implemented in tools ROCS [9] and ARCS [4]. The abstraction overhead can also be reduced by representing systems as a collection of components composed in parallel [6,13]. These have been developed in isolation and were not previously interoperable.

1.2 Methodology

Figure 1 depicts this paper's methodology and organization. The existing control synthesis formalism does not readily lend itself to algorithmic modifications that reflect and exploit structural properties in the system and specification. We use the theory of relational interfaces [22] as a foundation and augment it to express control synthesis pipelines. Interfaces are used to represent both system models and constraints. A small collection of atomic operators manipulates interfaces and is powerful enough to reconstruct many existing control synthesis pipelines.

One may also add new composite operators to encode desirable heuristics that exploit structural properties in the system and specifications. The last

three sections encode the techniques for abstraction-based control synthesis from Sect. 1.1 within the relational interfaces framework. By deliberately deconstructing those techniques, then reconstructing them within a compositional framework it was possible to identify implicit or unnecessary assumptions then generalize or remove them. It also makes the aforementioned techniques interoperable amongst themselves as well as future techniques.

Interfaces come equipped with a refinement partial order that formalizes when one interface abstracts another. This paper focuses on preserving the refinement relation and sufficient conditions to refine discrete controllers back to concrete ones. Additional guarantees regarding completeness, termination, precision, or decomposability can be encoded, but impose additional requirements on the control synthesis algorithm and are beyond the scope of this paper.

1.3 Contributions

To our knowledge, the application of relational interfaces to robust abstraction-based control synthesis is new. The framework's building blocks consist of a collection of small, well understood operators that are nonetheless powerful enough to express many prior techniques. Encoding these techniques as relational interface operations forced us to simplify, formalize, or remove implicit assumptions in existing tools. The framework also exhibits numerous desirable features.

1. It enables compositional tools for control synthesis by leveraging a theoretical foundation with compositionality built into it. This paper showcases a principled methodology to seamlessly combine the methods in Sect. 1.1, as well as construct new techniques.
2. It enables a declarative approach to control synthesis by enforcing a strict separation between the high level algorithm from its low level implementation. We rely on the availability of an underlying data structure to encode and manipulate predicates. Low level predicate operations, while powerful, make it easy to inadvertently violate the refinement property. Conforming to the relational interface operations minimizes this danger.

This paper's first half is domain agnostic and applicable to general robust control synthesis problems. The second half applies those insights to the finite abstraction approach to control synthesis. A smaller Dubins vehicle example is used to showcase and evaluate different techniques and their computational gains, compared to the unoptimized problem. In an extended version of this paper available at [1], a 6D lunar lander example leverages all techniques in this paper and introduces a few new ones.

1.4 Notation

Let $=$ be an *assertion* that two objects are mathematically equivalent; as a special case '$\equiv$' is used when those two objects are sets. In contrast, the operator '$==$' *checks* whether two objects are equivalent, returning true if they are and false otherwise. A special instance of '$==$' is logical equivalence '$\Leftrightarrow$'.

Variables are denoted by lower case letters. Each variable v is associated with a domain of values $\mathcal{D}(v)$ that is analogous to the variable's type. A composite variable is a set of variables and is analogous to a bundle of wrapped wires. From a collection of variables $v_1, \ldots, v_M$ a composite variable v can be constructed by taking the union $v \equiv v_1 \cup \ldots \cup v_M$ and the domain $\mathcal{D}(v) \equiv \prod_{i=1}^{M} \mathcal{D}(v_i)$. Note that the variables $v_1, \ldots, v_M$ above may themselves be composite. As an example if v is associated with a M-dimensional Euclidean space $\mathbb{R}^M$, then it is a composite variable that can be broken apart into a collection of atomic variables $v_1, \ldots, v_M$ where $\mathcal{D}(v_i) \equiv \mathbb{R}$ for all $i \in \{1, \ldots, M\}$. The technical results herein do not distinguish between composite and atomic variables.

Predicates are functions that map variable assignments to a Boolean value. Predicates that stand in for expressions/formulas are denoted with capital letters. Predicates P and Q are logically equivalent (denoted by $P \Leftrightarrow Q$) if and only if $P \Rightarrow Q$ and $Q \Rightarrow P$ are true for all variable assignments. The universal and existential quantifiers $\forall$ and $\exists$ eliminate variables and yield new predicates. Predicates $\exists w P$ and $\forall w P$ do not depend on w. If w is a composite variable $w \equiv w_1 \cup \ldots \cup w_N$ then $\exists w P$ is simply a shorthand for $\exists w_1 \ldots \exists w_N P$.

2 Control Synthesis for a Motivating Example

As a simple, instructive example consider a planar Dubins vehicle that is tasked with reaching a desired location. Let $x = \{p_x, p_y, \theta\}$ be the collection of state variables, $u = \{v, \omega\}$ be a collection input variables to be controlled, $x^+ = \{p_x^+, p_y^+, \theta^+\}$ represent state variables at a subsequent time step, and $L = 1.4$ be a constant representing the vehicle length. The constraints

$$p_x^+ == p_x + v\cos(\theta) \qquad (F_x)$$

$$p_y^+ == p_y + v\sin(\theta) \qquad (F_y)$$

$$\theta^+ == \theta + \frac{v}{L}\sin(\omega) \qquad (F_\theta)$$

characterize the discrete time dynamics. The continuous state domain is $\mathcal{D}(x) \equiv [-2, 2] \times [-2, 2] \times [-\pi, \pi)$, where the last component is periodic so $-\pi$ and π are identical values. The input domains are $\mathcal{D}(v) \equiv \{0.25, 0.5\}$ and $\mathcal{D}(\omega) \equiv \{-1.5, 0, 1.5\}$

Let predicate $F = F_x \wedge F_y \wedge F_\theta$ represent the monolithic system dynamics. Predicate T depends only on x and represents the target set $[-0.4, 0.4] \times [-0.4, 0.4] \times [-\pi, \pi)$, encoding that the vehicle's position must reach a square with any orientation. Let Z be a predicate that depends on variable x^+ that encodes a collection of states at a future time step. Equation (1) characterizes the robust controlled predecessor, which takes Z and computes the set of states from which there exists a non-blocking assignment to u that guarantees x^+ will satisfy Z, despite any non-determinism contained in F. The term $\exists x^+ F$ prevents state-control pairs from blocking, while $\forall x^+(F \Rightarrow Z)$ encodes the state-control pairs that guarantee satisfaction of Z.

$$\texttt{cpre}(F, Z) = \exists u(\exists x^+ F \wedge \forall x^+(F \Rightarrow Z)). \qquad (1)$$

The controlled predecessor is used to solve safety and reach games. We can solve for a region for which the target T (respectively, safe set S) can be reached (made invariant) via an iteration of an appropriate `reach` (`safe`) operator. Both iterations are given by:

$$\text{Reach Iter:} \quad Z_0 = \bot \qquad Z_{i+1} = \texttt{reach}(F, Z_i, T) = \texttt{cpre}(F, Z_i) \vee T. \tag{2}$$

$$\text{Safety Iter:} \quad Z_0 = S \qquad Z_{i+1} = \texttt{safe}(F, Z_i, S) = \texttt{cpre}(F, Z_i) \wedge S. \tag{3}$$

The above iterations are not guaranteed to reach a fixed point in a finite number of iterations, except under certain technical conditions [21]. Figure 2 depicts an approximate region where the controller can force the Dubins vehicle to enter T. We showcase different improvements relative to a base line script used to generate Fig. 2. A toolbox that adopts this paper's framework is being actively developed and is open sourced at [2]. It is written in `python 3.6` and uses the `dd` package as an interface to `CUDD` [20], a library in `C/C++` for constructing and manipulating binary decision diagrams (BDD). All experiments were run on a single core of a 2013 Macbook Pro with 2.4 GHz Intel Core i7 and 8 GB of RAM.

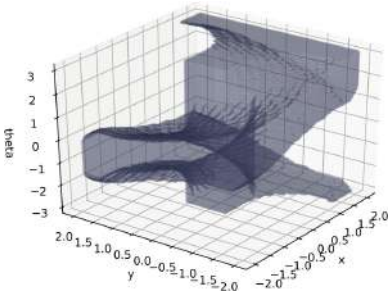


Fig. 2. Approximate solution to the Dubins vehicle reach game visualized as a subset of the state space.

The following section uses relational interfaces to represent the controlled predecessor `cpre`$(\cdot)$ and iterations (2) and (3) as a computational pipeline. Subsequent sections show how modifying this pipeline leads to favorable theoretical properties and computational gains.

3 Relational Interfaces

Relational interfaces are predicates augmented with annotations about each variable's role as an input or output[1]. They abstract away a component's internal implementation and only encode an input-output relation.

Definition 1 (Relational Interface [22]). *An interface $M(i, o)$ consists of a predicate M over a set of input variables i and output variables o.*

For an interface $M(i, o)$, we call (i, o) its input-output *signature*. An interface is a sink if it contains no outputs and has signature like $(i, \varnothing)$, and a source if it contains no inputs like $(\varnothing, o)$. Sinks and source interfaces can be interpreted as sets whereas input-output interfaces are relations. Interfaces encode relations through their predicates and can capture features such as non-deterministic outputs or

[1] Relational interfaces closely resemble assume-guarantee contracts [16]; we opt to use relational interfaces because inputs and outputs play a more prominent role.

blocking (i.e., disallowed, error) inputs. A system blocks for an input assignment if there does not exist a corresponding output assignment that satisfies the interface relation. Blocking is a critical property used to declare *requirements*; sink interfaces impose constraints by modeling constrain violations as blocking inputs. Outputs on the other hand exhibit non-determinism, which is treated as an *adversary*. When one interface's outputs are connected to another's inputs, the outputs seek to cause blocking whenever possible.

3.1 Atomic and Composite Operators

Operators are used to manipulate interfaces by taking interfaces and variables as inputs and yielding another interface. We will show how the controlled predecessor $\texttt{cpre}(\cdot)$ in (1) can be constructed by composing operators appearing in [22] and one additional one. The first, output hiding, removes interface outputs.

Definition 2 (Output Hiding [22]). *Output hiding operator* $\texttt{ohide}(w, F)$ *over interface* $F(i, o)$ *and outputs* w *yields an interface with signature* $(i, o \setminus w)$.

$$\texttt{ohide}(w, F) = \exists w F \tag{4}$$

Existentially quantifying out w ensures that the input-output behavior over the unhidden variables is still consistent with potential assignments to w. The operator $\texttt{nb}(\cdot)$ is a special variant of $\texttt{ohide}(\cdot)$ that hides all outputs, yielding a sink encoding all non-blocking inputs to the original interface.

Definition 3 (Nonblocking Inputs Sink). *Given an interface* $F(i, o)$*, the nonblocking operation* $\texttt{nb}(F)$ *yields a sink interface with signature* $(i, \varnothing)$ *and predicate* $\texttt{nb}(F) = \exists o F$*. If* $F(i, \varnothing)$ *is a sink interface, then* $\texttt{nb}(F) = F$ *yields itself. If* $F(\varnothing, o)$ *is a source interface, then* $\texttt{nb}(F) = \bot$ *if and only if* $F \Leftrightarrow \bot$*; otherwise* $\texttt{nb}(F) = \top$.

The interface composition operator takes multiple interfaces and "collapses" them into a single input-output interface. It can be viewed as a generalization of function composition in the special case where each interface encodes a total function (i.e., deterministic output and inputs never block).

Definition 4 (Interface Composition [22]). *Let* $F_1(i_1, o_1)$ *and* $F_2(i_2, o_2)$ *be interfaces with disjoint output variables* $o_1 \cap o_2 \equiv \varnothing$ *and* $i_1 \cap o_2 \equiv \varnothing$ *which signifies that* F_2*'s outputs may not be fed back into* F_1*'s inputs. Define new composite variables*

$$io_{12} \equiv o_1 \cap i_2 \tag{5}$$

$$i_{12} \equiv (i_1 \cup i_2) \setminus io_{12} \tag{6}$$

$$o_{12} \equiv o_1 \cup o_2. \tag{7}$$

Composition $\texttt{comp}(F_1, F_2)$ *is an interface with signature* (i_{12}, o_{12}) *and predicate*

$$F_1 \wedge F_2 \wedge \forall o_{12}(F_1 \Rightarrow \texttt{nb}(F_2)). \tag{8}$$

Interface subscripts may be swapped if instead F_2*'s outputs are fed into* F_1.

Interfaces F_1 and F_2 are composed in parallel if $io_{21} \equiv \varnothing$ holds in addition to $io_{12} \equiv \varnothing$. Equation (8) under parallel composition reduces to $F_1 \wedge F_2$ (Lemma 6.4 in [22]) and $\texttt{comp}(\cdot)$ is commutative and associative. If $io_{12} \not\equiv \varnothing$, then they are composed in series and the composition operator is only associative. Any acyclic interconnection can be composed into a single interface by systematically applying Definition 4's binary composition operator. Non-deterministic outputs are interpreted to be *adversarial*. Series composition of interfaces has a built-in notion of robustness to account for F_1's non-deterministic outputs and blocking inputs to F_2 over the shared variables io_{12}. The term $\forall o_{12}(F_1 \Rightarrow \texttt{nb}(F_2))$ in Eq. (8) is a predicate over the composition's input set i_{12}. It ensures that if a potential output of F_1 may cause F_2 to block, then $\texttt{comp}(F_1, F_2)$ must preemptively block.

The final atomic operator is input hiding, which may only be applied to sinks. If the sink is viewed as a constraint, an input variable is "hidden" by an angelic environment that chooses an input assignment to satisfy the constraint. This operator is analogous to projecting a set into a lower dimensional space.

Definition 5 (Hiding Sink Inputs). *Input hiding operator* $\texttt{ihide}(w, F)$ *over sink interface* $F(i, \varnothing)$ *and inputs* w *yields an interface with signature* $(i \setminus w, \varnothing)$.

$$\texttt{ihide}(w, F) = \exists w F \tag{9}$$

Unlike the composition and output hiding operators, this operator is not included in the standard theory of relational interfaces [22] and was added to encode a controller predecessor introduced subsequently in Eq. (10).

3.2 Constructing Control Synthesis Pipelines

The robust controlled predecessor (1) can be expressed through operator composition.

Proposition 1. *The controlled predecessor operator* (10) *yields a sink interface with signature* $(x, \varnothing)$ *and predicate equivalent to the predicate in* (1).

$$\texttt{cpre}(F, Z) = \texttt{ihide}(u, \texttt{ohide}(x^+, \texttt{comp}(F, Z))). \tag{10}$$

The simple proof is provided in the extended version at [1]. Proposition 1 signifies that controlled predecessors can be interpreted as an instance of robust composition of interfaces, followed by variable hiding. It can be shown that $\texttt{safe}(F, Z, S) = \texttt{comp}(\texttt{cpre}(F, Z), S)$ because $S(x, \varnothing)$ and $\texttt{cpre}(F, Z)$ would be composed in parallel.[2] Figure. 3 shows a visualization of the safety game's fixed point iteration from the point of view of relational interfaces. Starting from the right-most sink interface S (equivalent to Z_0) the iteration (3) constructs a sequence of sink interfaces $Z_1, Z_2, \ldots$ encoding relevant subsets of the state space. The numerous $S(x, \varnothing)$ interfaces impose constraints and can be interpreted as monitors that raise errors if the safety constraint is violated.

[2] Disjunctions over sinks are required to encode $\texttt{reach}(\cdot)$. This will be enabled by the shared refinement operator defined in Definition 10.

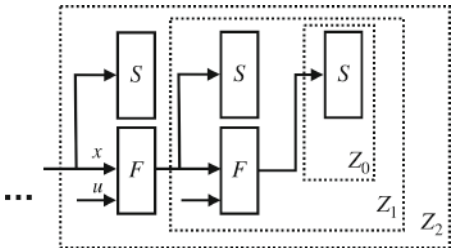


Fig. 3. Safety control synthesis iteration (3) depicted as a sequence of sink interfaces.

3.3 Modifying the Control Synthesis Pipeline

Equation (10)'s definition of $\texttt{cpre}(\cdot)$ is oblivious to the domains of variables x, u, and x^+. This generality is useful for describing a problem and serving as a blank template. Whenever problem structure exists, pipeline modifications refine the general algorithm into a form that reflects the specific problem instance. They also allow a user to inject implicit preferences into a problem and reduce computational bottlenecks or to refine a solution. The subsequent sections apply this philosophy to the abstraction-based control techniques from Sect. 1.1:

- Sect. 4: Coarsening interfaces reduces the computational complexity of a problem by throwing away fine grain information. The synthesis result is conservative but the degree of conservatism can be modified.
- Sect. 5: Refining interfaces decreases result conservatism. Refinement in combination with coarsening allows one to dynamically modulate the complexity of the problem as a function of multiple criteria such as the result granularity or minimizing computational resources.
- Sect. 6: If the dynamics or specifications are decomposable then the control predecessor operator can be broken apart to refect that decomposition.

These sections do more than simply reconstruct existing techniques in the language of relational interfaces. They uncover some implicit assumptions in existing tools and either remove them or make them explicit. Minimizing the number of assumptions ensures applicability to a diverse collection of systems and specifications and compatibility with future algorithmic modifications.

4 Interface Abstraction via Quantization

A key motivator behind abstraction-based control synthesis is that computing the game iterations from Eqs. (2) and (3) exactly is often intractable for high-dimensional nonlinear dynamics. Termination is also not guaranteed. Quantizing (or "abstracting") continuous interfaces into a finite counterpart ensures that each predicate operation of the game terminates in finite time but at the cost of the solution's precision. Finer quantization incurs a smaller loss of precision but

can cause the memory and computational requirements to store and manipulate the symbolic representation to exceed machine resources.

This section first introduces the notion of interface abstraction as a refinement relation. We define the notion of a quantizer and show how it is a simple generalization of many existing quantizers in the abstraction-based control literature. Finally, we show how one can inject these quantizers anywhere in the control synthesis pipeline to reduce computational bottlenecks.

4.1 Theory of Abstract Interfaces

While a controller synthesis algorithm can analyze a simpler model of the dynamics, the results have no meaning unless they can be extrapolated back to the original system dynamics. The following interface refinement condition formalizes a condition when this extrapolation can occur.

Definition 6 (Interface Refinement [22]). *Let $F(i, o)$ and $\hat{F}(\hat{i}, \hat{o})$ be interfaces. $\hat{F}$ is an abstraction of F if and only if $i \equiv \hat{i}$, $o \equiv \hat{o}$, and*

$$\mathbf{nb}(\hat{F}) \Rightarrow \mathbf{nb}(F) \tag{11}$$

$$\left(\mathbf{nb}(\hat{F}) \wedge F\right) \Rightarrow \hat{F} \tag{12}$$

are valid formulas. This relationship is denoted by $\hat{F} \preceq F$.

Definition 6 imposes two main requirements between a concrete and abstract interface. Equation (11) encodes the condition where if $\hat{F}$ accepts an input, then F must also accept it; that is, the abstract component is more aggressive with rejecting invalid inputs. Second, if both systems accept the input then the abstract output set is a superset of the concrete function's output set. The abstract interface is a conservative representation of the concrete interface because the abstraction accepts fewer inputs and exhibits more non-deterministic outputs. If both the interfaces are sink interfaces, then $\hat{F} \preceq F$ reduces down to $\hat{F} \subseteq F$ when $F, \hat{F}$ are interpreted as sets. If both are source interfaces then the set containment direction is flipped and $\hat{F} \preceq F$ reduces down to $F \subseteq \hat{F}$.

The refinement relation satisfies the required reflexivity, transitivity, and antisymmetry properties to be a partial order [22] and is depicted in Fig. 4. This order has a bottom element $\bot$ which is a universal abstraction. Conveniently, the bottom element $\bot$ signifies both boolean false and the bottom of the partial order. This interface blocks for every potential input. In contrast, Boolean $\top$ plays no special role in the partial order. While $\top$ exhibits totally non-deterministic outputs, it also accepts inputs. A blocking input is considered "worse" than non-deterministic outputs in the refinement order. The refinement relation $\preceq$ encodes a direction of conservatism such that any reasoning done over the abstract models is sound and can be generalized to the concrete model.

Theorem 1 (Informal Substitutability Result [22]). *For any input that is allowed for the abstract model, the output behaviors exhibited by an abstract model contains the output behaviors exhibited by the concrete model.*

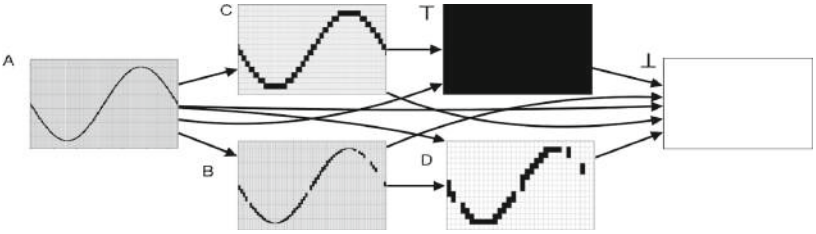


Fig. 4. Example depiction of the refinement partial order. Each small plot on the depicts input-output pairs that satisfy an interface's predicate. Inputs (outputs) vary along the horizontal (vertical) axis. Because B blocks on some inputs but A accepts all inputs $B \preceq A$. Interface C exhibits more output non-determinism than A so $C \preceq A$. Similarly $D \preceq B$, $D \preceq C$, $\top \preceq C$, etc. Note that B and C are incomparable because C exhibits more output non-determinism and B blocks for more inputs. The false interface $\bot$ is a universal abstraction, while $\top$ is incomparable with B and D.

If a property on outputs has been established for an abstract interface, then it still holds if the abstract interface is replaced with the concrete one. Informally, the abstract interface is more conservative so if a property holds with the abstraction then it must also hold for the true system. All aforementioned interface operators preserve the properties of the refinement relation of Definition 6, in the sense that they are monotone with respect to the refinement partial order.

Theorem 2 (Composition Preserves Refinement [22]). *Let* $\hat{A} \preceq A$ *and* $\hat{B} \preceq B$*. If the composition is well defined, then* $\texttt{comp}(\hat{A}, \hat{B}) \preceq \texttt{comp}(A, B)$*.*

Theorem 3 (Output Hiding Preserves Refinement [22]). *If* $A \preceq B$*, then* $\texttt{ohide}(w, A) \preceq \texttt{ohide}(w, B)$ *for any variable* w*.*

Theorem 4 (Input Hiding Preserves Refinement). *If* A, B *are both sink interfaces and* $A \preceq B$*, then* $\texttt{ihide}(w, A) \preceq \texttt{ihide}(w, B)$ *for any variable* w*.*

Proofs for Theorems 2 and 3 are provided in [22]. Theorem 4's proof is simple and is omitted. One can think of using interface composition and variable hiding to horizontally (with respect to the refinement order) navigate the space of all interfaces. The synthesis pipeline encodes one navigated path and monotonicity of these operators yields guarantees about the path's end point. Composite operators such as $\texttt{cpre}(\cdot)$ chain together multiple incremental steps. Furthermore since the composition of monotone operators is itself a monotone operator, any composite constructed from these parts is also monotone. In contrast, the coarsening and refinement operators introduced later in Definitions 8 and 10 respectively are used to move vertically and construct abstractions. The "direction" of new composite operators can easily be established through simple reasoning about the cumulative directions of their constituent operators.

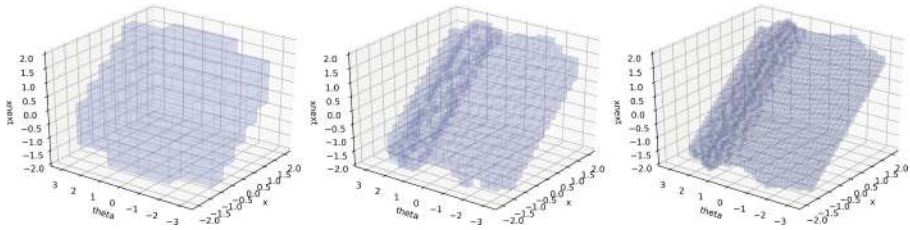


Fig. 5. Coarsening of the F_x interface to $2^3, 2^4$ and 2^5 bins along each dimension for a fixed v assignment. Interfaces are coarsened within milliseconds for BDDs but the runtime depends on the finite abstraction's data structure representation.

4.2 Dynamically Coarsening Interfaces

In practice, the sequence of interfaces Z_i generated during synthesis grows in complexity. This occurs even if the dynamics F and the target/safe sets have compact representations (i.e., fewer nodes if using BDDs). Coarsening F and Z_i combats this growth in complexity by effectively reducing the amount of information sent between iterations of the fixed point procedure.

Spatial discretization or *coarsening* is achieved by use of a quantizer interface that implicitly aggregates points in a space into a partition or cover.

Definition 7. *A quantizer $Q(i, o)$ is any interface that abstracts the identity interface $(i == o)$ associated with the signature (i, o).*

Quantizers decrease the complexity of the system representation and make synthesis more computationally tractable. A coarsening operator abstracts an interface by connecting it in series with a quantizer. Coarsening reduces the number of non-blocking inputs and increases the output non-determinism.

Definition 8 (Input/Output Coarsening). *Given an interface $F(i, o)$ and input quantizer $Q(\hat{i}, i)$, input coarsening yields an interface with signature $(\hat{i}, o)$.*

$$\texttt{icoarsen}(F, Q(\hat{i}, i)) = \texttt{ohide}(i, \texttt{comp}(Q(\hat{i}, i), F)) \tag{13}$$

Similarly, given an output quantizer $Q(o, \hat{o})$, output coarsening yields an interface with signature $(i, \hat{o})$.

$$\texttt{ocoarsen}(F, Q(o, \hat{o})) = \texttt{ohide}(o, \texttt{comp}(F, Q(o, \hat{o}))) \tag{14}$$

Figure 5 depicts how coarsening reduces the information required to encode a finite interface. It leverages a variable precision quantizer, whose implementation is described in the extended version at [1].

The corollary below shows that quantizers can be seamlessly integrated into the synthesis pipeline while preserving the refinement order. It readily follows from Theorems 2, 3, and the quantizer definition.

Corollary 1. *Input and output coarsening operations (13) and (14) are monotone operations with respect to the interface refinement order $\preceq$.*

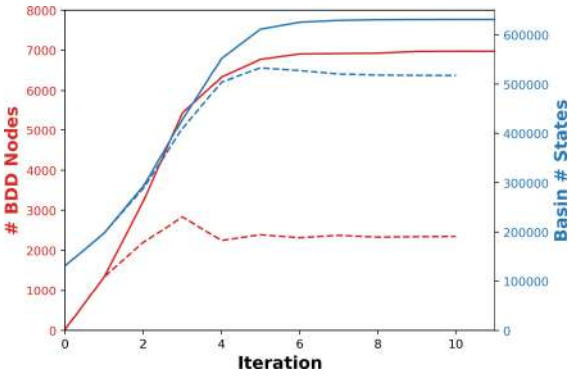


Fig. 6. Number of BDD nodes (red) and number of states in reach basin (blue) with respect to the reach game iteration with a greedy quantization. The solid lines result from the unmodified game with no coarsening heuristic. The dashed lines result from greedy coarsening whenever the winning region exceeds 3000 BDD nodes. (Color figure online)

It is difficult to know a priori where a specific problem instance lies along the spectrum between mathematical precision and computational efficiency. It is then desirable to coarsen dynamically in response to runtime conditions rather than statically beforehand. Coarsening heuristics for reach games include:

- *Downsampling with progress* [7]: Initially use coarser system dynamics to rapidly identify a coarse reach basin. Finer dynamics are used to construct a more granular set whenever the coarse iteration "stalls". In [7] only the Z_i are coarsened during synthesis. We enable the dynamics F to be as well.
- *Greedy quantization*: Selectively coarsening along certain dimensions by checking at runtime which dimension, when coarsened, would cause Z_i to shrink the least. This reward function can be leveraged in practice because coarsening is computationally cheaper than composition. For BDDs, the winning region can be coarsened until the number of nodes reduces below a desired threshold. Figure 6 shows this heuristic being applied to reduce memory usage at the expense of answer fidelity. A fixed point is not guaranteed as long as quantizers can be dynamically inserted into the synthesis pipeline, but is once quantizers are always inserted at a fixed precision.

The most common quantizer in the literature never blocks and only increases non-determinism (such quantizers are called "strict" in [18,19]). If a quantizer is interpreted as a partition or cover, this requirement means that the union must be equal to an entire space. Definition 7 relaxes that requirement so the union can be a subset instead. It also hints at other variants such as interfaces that don't increase output non-determinism but instead block for more inputs.

5 Refining System Dynamics

Shared refinement [22] is an operation that takes two interfaces and merges them into a single interface. In contrast to coarsening, it makes interfaces more precise. Many tools construct system abstractions by starting from the universal abstraction $\perp$, then iteratively refining it with a collection of smaller interfaces that represent input-output samples. This approach is especially useful if the canonical concrete system is a black box function, Simulink model, or source code file. These representations do not readily lend themselves to the predicate operations or be coarsened directly. We will describe later how other tools implement a restrictive form of refinement that introduces unnecessary dependencies.

Interfaces can be successfully merged whenever they do not contain contradictory information. The shared refinability condition below formalizes when such a contradiction does not exist.

Definition 9 (Shared Refinability [22]). *Interfaces $F_1(i, o)$ and $F_2(i, o)$ with identical signatures are shared refinable if*

$$(\mathit{nb}(F_1) \wedge \mathit{nb}(F_2)) \Rightarrow \exists o(F_1 \wedge F_2) \tag{15}$$

For any inputs that do not block for all interfaces, the corresponding output sets must have a non-empty intersection. If multiple shared refinable interfaces, then they can be combined into a single one that encapsulates all of their information.

Definition 10 (Shared Refinement Operation [22]). *The shared refinement operation combines two shared refinable interfaces F_1 and F_2, yielding a new identical signature interface corresponding to the predicate*

$$\mathit{refine}(F_1, F_2) = (\mathit{nb}(F_1) \vee \mathit{nb}(F_2)) \wedge (\mathit{nb}(F_1) \Rightarrow F_1) \wedge (\mathit{nb}(F_2) \Rightarrow F_2). \tag{16}$$

The left term expands the set of accepted inputs. The right term signifies that if an input was accepted by multiple interfaces, the output must be consistent with each of them. The shared refinement operation reduces to disjunction for sink interfaces and to conjunction for source interfaces.

Shared refinement's effect is to move up the refinement order by combining interfaces. Given a collection of shared refinable interfaces, the shared refinement operation yields the least upper bound with respect to the refinement partial order in Definition 6. Violation of (15) can be detected if the interfaces fed into `refine`$(\cdot)$ are not abstractions of the resulting interface.

5.1 Constructing Finite Interfaces Through Shared Refinement

A common method to construct finite abstractions is through simulation and overapproximation of forward reachable sets. This technique appears in tools such as PESSOA [12], SCOTS [19], MASCOT [7], ROCS [9] and ARCS [4]. By covering a sufficiently large portion of the interface input space, one can construct larger composite interfaces from smaller ones via shared refinement.

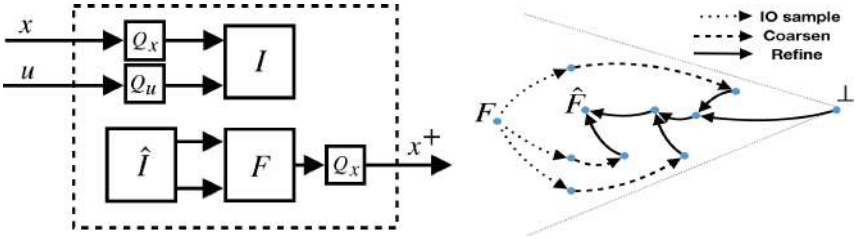


Fig. 7. (Left) Result of sample and coarsen operations for control system interface $F(x \cup u, x^+)$. The I and $\hat{I}$ interfaces encode the same predicate, but play different roles as sink and source. (Right) Visualization of finite abstraction as traversing the refinement partial order. Nodes represent interfaces and edges signify data dependencies for interface manipulation operators. Multiple refine edges point to a single node because refinement combines multiple interfaces. Input-output (IO) sample and coarsening are unary operations so the resulting nodes only have one incoming edge. The concrete interface F refines all others, and the final result is an abstraction $\hat{F}$.

Smaller interfaces are constructed by sampling regions of the input space and constructing an input-output pair. In Fig. 7's left half, a sink interface $I(x \cup u, \varnothing)$ acts as a filter. The source interface $\hat{I}(\varnothing, x \cup u)$ composed with $F(x \cup u, x^+)$ prunes any information that is outside the relevant input region. The original interface refines any sampled interface. To make samples *finite*, interface inputs and outputs are coarsened. An individual sampled abstraction is not useful for synthesis because it is restricted to a local portion of the interface input space. After sampling many finite interfaces are merged through shared refinement. The assumption $\hat{I}_i \Rightarrow \texttt{nb}(F)$ encodes that the dynamics won't raise an error when simulated and is often made implicitly. Figure 7's right half depicts the sample, coarsen, and refine operations as methods to vertically traverse the interface refinement order.

Critically, `refine`$(\cdot)$ can be called within the synthesis pipeline and does not assume that the sampled interfaces are disjoint. Figure 8 shows the results from refining the dynamics with a collection of state-control hyper-rectangles that are randomly generated via uniformly sampling their widths and offsets along each dimension. These hyper-rectangles may overlap. If the same collection of hyper-rectangles were used in MASCOT, SCOTS, ARCS, or ROCS then this would yield a much more conservative abstraction of the dynamics because their implementations are not robust to overlapping or misaligned samples. PESSOA and SCOTS circumvent this issue altogether by enforcing disjointness with an exhaustive traversal of the state-control space, at the cost of unnecessarily coupling the refinement and sampling procedures. The lunar lander in the extended version [1] embraces overlapping and uses two mis-aligned grids to construct a grid partition with p^N elements with only $p^N(\frac{1}{2})^{N-1}$ samples (where p is the number of bins along each dimension and N is the interface input dimension). This technique introduces a small degree of conservatism but its computational savings typically outweigh this cost.

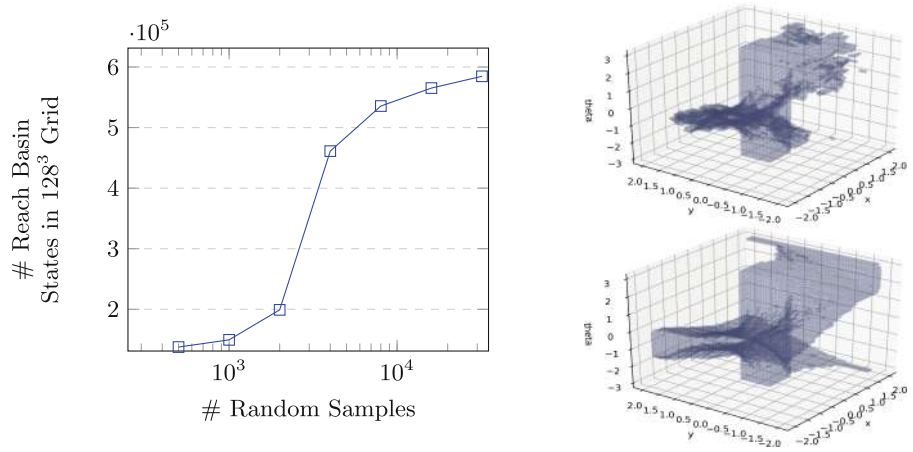


Fig. 8. The number of states in the computed reach basin grows with the number of random samples. The vertical axis is lower bounded by the number of states in the target $131k$ and upper bounded by $631k$, the number of states using an exhaustive traversal. Naive implementations of the exhaustive traversal would require 12 million samples. The right shows basins for 3000 (top) and 6000 samples (bottom).

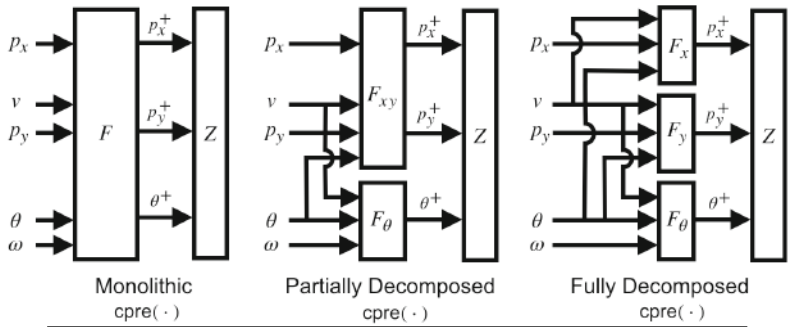
6 Decomposed Control Predecessor

A decomposed control predecessor is available whenever the system state space consists of a Cartesian product and the dynamics are decomposed component-wise such as F_x, F_y, and F_θ for the Dubins vehicle. This property is common for continuous control systems over Euclidean spaces. While one may construct F directly via the abstraction sampling approach, it is often intractable for larger dimensional systems. A more sophisticated approach abstracts the lower dimensional components F_x, F_y, and F_θ individually, computes $F = \texttt{comp}(F_x, F_y, F_\theta)$, then feeds it to the monolithic $\texttt{cpre}(\cdot)$ from Proposition 1. This section's approach is to avoid computing F at all and decompose the monolithic $\texttt{cpre}(\cdot)$. It operates by breaking apart the term $\textit{ohide}(x^+, \texttt{comp}(F, Z))$ in such a way that it respects the decomposition structure. For the Dubins vehicle example $\textit{ohide}(x^+, \texttt{comp}(F, Z))$ is replaced with

$$\textit{ohide}(p_x^+, \texttt{comp}(F_x, \textit{ohide}(p_y^+, \texttt{comp}(F_y, \textit{ohide}(\theta^+, \texttt{comp}(F_\theta, Z))))))$$

yielding a sink interface with inputs p_x, p_y, v, θ, and ω. This representation and the original $\textit{ohide}(x^+, \texttt{comp}(F, Z))$ are equivalent because $\texttt{comp}(\cdot)$ is associative and interfaces do not share outputs $x^+ \equiv \{p_x^+, p_y^+, \theta^+\}$. Figure 9 shows multiple variants of $\texttt{cpre}(\cdot)$ and improved runtimes when one avoids preemptively constructing the monolithic interface. The decomposed $\texttt{cpre}(\cdot)$ resembles techniques to exploit partitioned transition relations in symbolic model checking [5].

No tools from Sect. 1.1 natively support decomposed control predecessors. We've shown a decomposed abstraction for components composed in parallel



Decomposition	Parallel Compose Runtime (s)	Reach Game Runtime (s)
F (Monolithic)	0.56	103.09
$F_{y\theta}, F_x$ (Partially Decomp.)	0.02	28.31
$F_{x\theta}, F_y$ (Partially Decomp.)	0.01	28.71
F_{xy}, F_θ (Partially Decomp.)	0.06	10.61
F_x, F_y, F_θ (Fully Decomp.)	n/a	4.42

Fig. 9. A monolithic `cpre`$(\cdot)$ incurs unnecessary pre-processing and synthesis runtime costs for the Dubins vehicle reach game. Each variant of `cpre`$(\cdot)$ above composes the interfaces F_x, F_y and F_θ in different permutations. For example, F_{xy} represents `comp`(F_x, F_y) and F represents `comp`(F_x, F_y, F_θ).

but this can also be generalized to series composition to capture, for example, a system where multiple components have different temporal sampling periods.

7 Conclusion

Tackling difficult control synthesis problems will require exploiting *all* available structure in a system with tools that can *flexibly adapt* to an individual problem's idiosyncrasies. This paper lays a foundation for developing an extensible suite of interoperable techniques and demonstrates the potential computational gains in an application to controller synthesis with finite abstractions. Adhering to a simple yet powerful set of well-understood primitives also constitutes a disciplined methodology for algorithm development, which is especially necessary if one wants to develop concurrent or distributed algorithms for synthesis.

References

1. http://arxiv.org/abs/1905.09503
2. https://github.com/ericskim/redax/tree/CAV19
3. Basar, T., Olsder, G.J.: Dynamic Noncooperative Game Theory, vol. 23. Siam, Philadelphia (1999)

4. Bulancea, O.L., Nilsson, P., Ozay, N.: Nonuniform abstractions, refinement and controller synthesis with novel BDD encodings. CoRR, arXiv: abs/1804.04280 (2018)
5. Burch, J., Clarke, E., Long, D.: Symbolic model checking with partitioned transition relations (1991)
6. Gruber, F., Kim, E., Arcak, M.: Sparsity-aware finite abstraction. In: 2017 IEEE 56th Conference on Decision and Control (CDC), December 2017
7. Hsu, K., Majumdar, R., Mallik, K., Schmuck, A.-K.: Multi-layered abstraction-based controller synthesis for continuous-time systems. In: Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week), HSCC 2018, pp. 120–129. ACM, New York (2018)
8. Khalil, H.K., Grizzle, J.W.: Nonlinear Systems, vol. 3. Prentice Hall, Upper Saddle River, New Jersey (2002)
9. Li, Y., Liu, J.: ROCS: a robustly complete control synthesis tool for nonlinear dynamical systems. In: Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week), HSCC 2018, pp. 130–135. ACM, New York (2018)
10. Liu, J.: Robust abstractions for control synthesis: completeness via robustness for linear-time properties. In: Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017, pp. 101–110. ACM, New York (2017)
11. Majumdar, R.: Symbolic algorithms for verification and control. Ph.D. thesis, University of California, Berkeley (2003)
12. Mazo Jr., M., Davitian, A., Tabuada, P.: PESSOA: a tool for embedded controller synthesis. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 566–569. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_49
13. Meyer, P.J., Girard, A., Witrant, E.: Compositional abstraction and safety synthesis using overlapping symbolic models. IEEE Trans. Autom. Control. **63**, 1835–1841 (2017)
14. Mouelhi, S., Girard, A., Gössler, G.: CoSyMA: a tool for controller synthesis using multi-scale abstractions. In: 16th International Conference on Hybrid Systems: Computation and Control, pp. 83–88. ACM (2013)
15. Nilsson, P., Ozay, N., Liu, J.: Augmented finite transition systems as abstractions for control synthesis. Discret. Event Dyn. Syst. **27**(2), 301–340 (2017)
16. Nuzzo, P.: Compositional design of cyber-physical systems using contracts. Ph.D. thesis, EECS Department, University of California, Berkeley, August 2015
17. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2005). https://doi.org/10.1007/11609773_24
18. Reißig, G., Weber, A., Rungger, M.: Feedback refinement relations for the synthesis of symbolic controllers. IEEE Trans. Autom. Control. **62**(4), 1781–1796 (2017)
19. Rungger, M., Zamani, M.: SCOTS: a tool for the synthesis of symbolic controllers. In: 19th International Conference on Hybrid Systems: Computation and Control, pp. 99–104. ACM (2016)
20. Somenzi, F.: CUDD: CU Decision Diagram Package. http://vlsi.colorado.edu/~fabio/CUDD/, Version 3.0.0 (2015)
21. Tabuada, P.: Verification and Control of Hybrid Systems. Springer, New York (2009). https://doi.org/10.1007/978-1-4419-0224-5

22. Tripakis, S., Lickly, B., Henzinger, T.A., Lee, E.A.: A theory of synchronous relational interfaces. ACM Trans. Program. Lang. Syst. (TOPLAS) **33**(4), 14 (2011)
23. Zamani, M., Pola, G., Mazo, M., Tabuada, P.: Symbolic models for nonlinear control systems without stability assumptions. IEEE Trans. Autom. Control **57**(7), 1804–1809 (2012)

Temporal Stream Logic: Synthesis Beyond the Bools

Bernd Finkbeiner[1], Felix Klein[1(✉)], Ruzica Piskac[2], and Mark Santolucito[2]

[1] Saarland University, Saarbrücken, Germany
klein@react.uni-saarland.de
[2] Yale University, New Haven, USA

Abstract. Reactive systems that operate in environments with complex data, such as mobile apps or embedded controllers with many sensors, are difficult to synthesize. Synthesis tools usually fail for such systems because the state space resulting from the discretization of the data is too large. We introduce TSL, a new temporal logic that separates control and data. We provide a CEGAR-based synthesis approach for the construction of implementations that are guaranteed to satisfy a TSL specification for all possible instantiations of the data processing functions. TSL provides an attractive trade-off for synthesis. On the one hand, synthesis from TSL, unlike synthesis from standard temporal logics, is undecidable in general. On the other hand, however, synthesis from TSL is scalable, because it is independent of the complexity of the handled data. Among other benchmarks, we have successfully synthesized a music player Android app and a controller for an autonomous vehicle in the Open Race Car Simulator (TORCS).

1 Introduction

In reactive synthesis, we automatically translate a formal specification, typically given in a temporal logic, into a controller that is guaranteed to satisfy the specification. Over the past two decades there has been much progress on reactive synthesis, both in terms of algorithms, notably with techniques like GR(1)-synthesis [7] and bounded synthesis [20], and in terms of tools, as showcased, for example, in the annual SYNTCOMP competition [25].

In practice however, reactive synthesis has seen limited success. One of the largest published success stories [6] is the synthesis of the AMBA bus protocol. To push synthesis even further, automatically synthesizing a controller for

Supported by the European Research Council (ERC) Grant OSARES (No. 683300), the German Research Foundation (DFG) as part of the Collaborative Research Center Foundations of Perspicuous Software Systems (TRR 248, 389792660), and the National Science Foundation (NSF) Grant CCF-1302327.

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 609–629, 2019.
https://doi.org/10.1007/978-3-030-25540-4_35

an autonomous system has been recognized to be of critical importance [52]. Despite many years of experience with synthesis tools, our own attempts to synthesize such controllers with existing tools have been unsuccessful. The reason is that the tools are unable to handle the data complexity of the controllers. The controller only needs to switch between a small number of behaviors, like steering during a bend, or shifting gears on high rpm. The number of control states in a typical controller (cf. [18]) is thus not much different from the arbiter in the AMBA case study. However, in order to correctly initiate transitions between control states, the driving controller must continuously process data from more than 20 sensors.

If this data is included (even as a rough discretization) in the state space of the controller, then the synthesis problem is much too large to be handled by any available tools. It seems clear then, that a scalable synthesis approach must separate control and data. If we assume that the data processing is handled by some other approach (such as deductive synthesis [38] or manual programming), is it then possible to solve the remaining reactive synthesis problem?

In this paper, we show scalable reactive synthesis is indeed possible. Separating data and control has allowed us to synthesize reactive systems, including an autonomous driving controller and a music player app, that had been impossible to synthesize with previously available tools. However, the separation of data and control implies some fundamental changes to reactive synthesis, which we describe in the rest of the paper. The changes also imply that the reactive synthesis problem is no longer, in general, decidable. We thus trade theoretical decidability for practical scalability, which is, at least with regard to the goal of synthesizing realistic systems, an attractive trade-off.

We introduce Temporal Stream Logic (TSL), a new temporal logic that includes *updates*, such as $[\![\mathtt{y} \leftarrowtail \mathtt{f\ x}]\!]$, and predicates over arbitrary function terms. The update $[\![\mathtt{y} \leftarrowtail \mathtt{f\ x}]\!]$ indicates that the result of applying function `f` to variable `x` is assigned to `y`. The implementation of predicates and functions is not part of the synthesis problem. Instead, we look for a system that satisfies the TSL specification *for all possible interpretations of the functions and predicates.*

This implicit quantification over all possible interpretations provides a useful abstraction: it allows us to *independently* implement the data processing part. On the other hand, this quantification is also the reason for the undecidability of the synthesis problem. If a predicate is applied to the same term *twice*, it must (independently of the interpretation) return the *same* truth value. The synthesis must then implicitly maintain a (potentially infinite) set of terms to which the predicate has previously been applied. As we show later, this set of terms can be used to encode PCP [45] for a proof of undecidability.

We present a practical synthesis approach for TSL specifications, which is based on bounded synthesis [20] and counterexample-guided abstraction refinement (CEGAR) [9]. We use bounded synthesis to search for an implementation up to a (iteratively growing) bound on the number of states. This approach underapproximates the actual TSL synthesis problem by leaving the interpretation of the predicates to the environment. The underapproximation allows

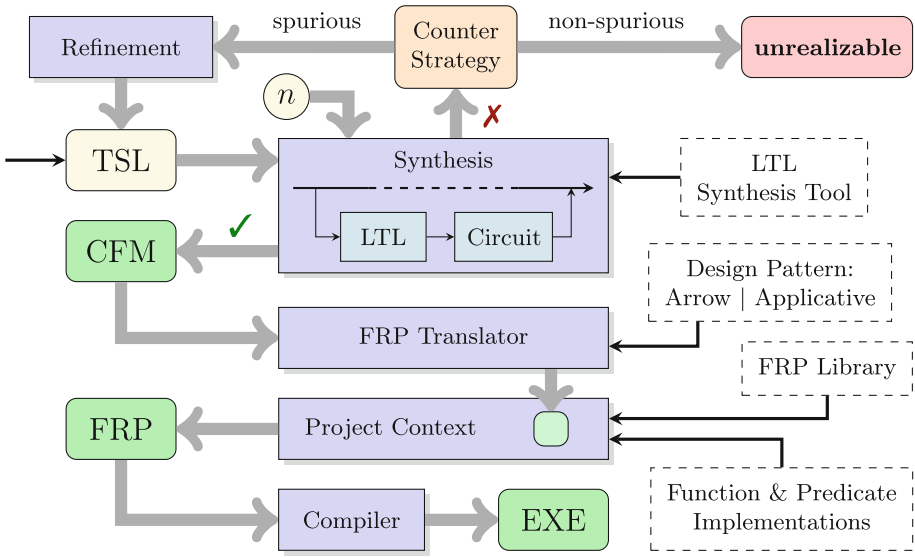


Fig. 1. The TSL synthesis procedure uses a modular design. Each step takes input from the previous step as well as interchangeable modules (dashed boxes).

for inconsistent behaviors: the environment might assign different truth values to the same predicate when evaluated at different points in time, even if the predicate is applied to the same term. However, if we find an implementation in this underapproximation, then the CEGAR loop terminates and we have a correct implementation for the original TSL specification. If we do not find an implementation in the underapproximation, we compute a counter strategy for the environment. Because bounded synthesis reduces the synthesis problem to a safety game, the counter strategy is a reachability strategy that can be represented as a finite tree. We check whether the counter strategy is spurious by searching for a pair of positions in the strategy where some predicate results in different truth values when applied to the same term. If the counter strategy is not spurious, then no implementation exists for the considered bound, and we increase the bound. If the counter strategy is spurious, then we introduce a constraint into the specification that eliminates the incorrect interpretation of the predicate, and continue with the refined specification.

A general overview of this procedure is shown in Fig. 1. The top half of the figure depicts the bounded search for an implementation that realizes a TSL specification using the CEGAR loop to refine the specification. If the specification is realizable, we proceed in the bottom half of the process, where a synthesized implementation is converted to a control flow model (CFM) determining the control of the system. We then specialize the CFM to Functional Reactive Programming (FRP), which is a popular and expressive programming paradigm for building reactive programs using functional programming languages [14].

```
Sys.leaveApp() :
  if (MP.musicPlaying())
    Ctrl.pause()
Sys.resumeApp() :
  pos = MP.trackPos()
  Ctrl.play(Tr,pos)
```

$$\texttt{ALWAYS}\ \big(\texttt{leaveApp Sys} \wedge \texttt{musicPlaying MP} \rightarrow [\![\texttt{Ctrl} \leftarrowtail \texttt{pause}()]\!]\big)$$

$$\texttt{ALWAYS}\ \big(\texttt{resumeApp Sys} \rightarrow [\![\texttt{Ctrl} \leftarrowtail \texttt{play Tr (trackPos MP)}]\!]\big)$$

Fig. 2. Sample code and specification for the music player app.

Our framework supports any FRP library using the *Arrow* or *Applicative* design patterns, which covers most of the existing FRP libraries (e.g. [2,3,10,41]). Finally, the synthesized control flow is embedded into a project context, where it is equipped with function and predicate implementations and then compiled to an executable program.

Our experience with synthesizing systems based on TSL specifications has been extremely positive. The synthesis works for a broad range of benchmarks, ranging from classic reactive synthesis problems (like escalator control), through programming exercises from functional reactive programming, to novel case studies like our music player app and the autonomous driving controller for a vehicle in the Open Race Car Simulator (TORCS).

2 Motivating Example

To demonstrate the utility of our method, we synthesized a music player Android app[1] from a TSL specification. A major challenge in developing Android apps is the temporal behavior of an app through the *Android lifecycle* [46]. The Android lifecycle describes how an app should handle being paused, when moved to the background, coming back into focus, or being terminated. In particular, *resume and restart errors* are commonplace and difficult to detect and correct [46]. Our music player app demonstrates a situation in which a resume and restart error could be unwittingly introduced when programming by hand, but is avoided by providing a specification. We only highlight the key parts of the example here to give an intuition of TSL. The complete specification is presented in [19].

Our music player app utilizes the Android music player library (`MP`), as well as its control interface (`Ctrl`). It pauses any playing music when moved to the background (for instance if a call is received), and continues playing the currently selected track (`Tr`) at the last track position when the app is resumed. In the Android system (`Sys`), the `leaveApp` method is called whenever the app moves to the background, while the `resumeApp` method is called when the app is brought back to the foreground. To avoid confusion between pausing music and pausing the app, we use `leaveApp` and `resumeApp` in place of the Android methods

[1] https://play.google.com/store/apps/details?id=com.mark.myapplication.

```
bool wasPlaying = false

Sys.leaveApp() :
  if (MP.musicPlaying()) :
    wasPlaying = true
    Ctrl.pause()
  else
    wasPlaying = false

Sys.resumeApp() :
  if (wasPlaying)
    pos = MP.trackPos()
    Ctrl.play(Tr,pos)
```

```
ALWAYS ((leaveApp Sys ∧ musicPlaying MP
        → ⟦Ctrl ↼ pause()⟧)
        ∧ (⟦Ctrl ↼ play Tr (trackPos MP)⟧
          AS_SOON_AS resumeApp Sys))
```

Fig. 3. The effect of a minor change in functionality on code versus a specification.

onPause and onResume. A programmer might manually write code for this as shown on the left in Fig. 2.

The behavior of this can be directly described in TSL as shown on the right in Fig. 2. Even eliding a formal introduction of the notation for now, the specification closely matches the textual specification. First, when the user leaves the app and the music is playing, the music pauses. Likewise for the second part, when the user resumes the app, the music starts playing again.

However, assume we want to change the behavior so that the music only plays on resume when the music had been playing before leaving the app in the first place. In the manually written program, this new functionality requires an additional variable `wasPlaying` to keep track of the music state. Managing the state requires multiple changes in the code as shown on the left in Fig. 3. The required code changes include: a conditional in the `resumeApp` method, setting `wasPlaying` appropriately in two places in `leaveApp`, and providing an initial value. Although a small example, it demonstrates how a minor change in functionality may require wide-reaching code changes. In addition, this change introduces a globally scoped variable, which then might accidentally be set or read elsewhere. In contrast, it is a simple matter to change the TSL specification to reflect this new functionality. Here, we only update one part of the specification to say that if the user leaves the app and the music is playing, the music has to play again as soon as the app resumes.

Synthesis allows us to specify a temporal behavior without worrying about the implementation details. In this example, writing the specification in TSL has eliminated the need of an additional state variable, similarly to a higher order `map` eliminating the need for an iteration variable. However, in more complex examples the benefits compound, as TSL provides a modular interface to specify behaviors, offloading the management of multiple interconnected temporal behaviors from the user to the synthesis engine.

3 Preliminaries

We assume time to be discrete and denote it by the set $\mathbb{N}$ of positive integers. A value is an arbitrary object of arbitrary type. $\mathcal{V}$ denotes the set of all values. The Boolean values are denoted by $\mathcal{B} \subseteq \mathcal{V}$. A stream $s\colon \mathbb{N} \to \mathcal{V}$ is a function fixing values at each point in time. An n-ary function $f\colon \mathcal{V}^n \to \mathcal{V}$ determines new values from n given values, where the set of all functions (of arbitrary arity) is given by $\mathcal{F}$. Constants are functions of arity 0. Every constant is a value, i.e., is an element of $\mathcal{F} \cap \mathcal{V}$. An n-ary predicate $p\colon \mathcal{V}^n \to \mathcal{B}$ checks a property over n values. The set of all predicates (of arbitrary arity) is given by $\mathcal{P}$, where $\mathcal{P} \subseteq \mathcal{F}$. We use $B^{[A]}$ to denote the set of all total functions with domain A and image B.

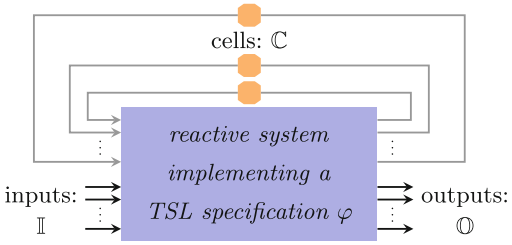
In the classical synthesis setting, inputs and outputs are vectors of Booleans, where the standard abstraction treats inputs and outputs as atomic propositions $\mathcal{I} \cup \mathcal{O}$, while their Boolean combinations form an alphabet $\Sigma = 2^{\mathcal{I} \cup \mathcal{O}}$. Behavior then is described through infinite sequences $\alpha = \alpha(0)\alpha(1)\alpha(2)\ldots \in \Sigma^\omega$. A *specification* describes a relation between input sequences $\alpha \in (2^{\mathcal{I}})^\omega$ and output sequences $\beta \in (2^{\mathcal{O}})^\omega$. Usually, this relation is not given by explicit sequences, but by a fomula in a temporal logic. The most popular such logic is Linear Temporal Logic (LTL) [43], which uses Boolean connectives to specify behavior at specific points in time, and temporal connectives, to relate sub-specifications over time. The realizability and synthesis problems for LTL are 2ExpTime-complete [44].

An implementation describes a realizing strategy, formalized via infinite trees. A Φ-labeled and Υ-branching tree is a function $\sigma\colon \Upsilon^* \to \Phi$, where Υ denotes the set of branching directions along a tree. Every node of the tree is given by a finite prefix $v \in \Upsilon^*$, which fixes the path to reach a node from the root. Every node is labeled by an element of Φ. For infinite paths $\nu \in \Upsilon^\omega$, the branch $\sigma\langle\nu$ denotes the sequence of labels that appear on ν, i.e., $\forall t \in \mathbb{N}.\ (\sigma\langle\nu)(t) = \sigma(\nu(0)\ldots\nu(t-1))$.

4 Temporal Stream Logic

We present a new logic: Temporal Stream Logic (TSL), which is especially designed for synthesis and allows for the manipulation of infinite streams of arbitrary (even non-enumerative, or higher order) type. It provides a straightforward notation to specify how outputs are computed from inputs, while using an intuitive interface to access time. The main focus of TSL is to describe temporal control flow, while abstracting away concrete implementation details. This not only keeps the logic intuitive and simple, but also allows a user to identify problems in the control flow even without a concrete implementation at hand. In this way, the use of TSL scales up to any required abstraction, such as API calls or complex algorithmic transformations.

Architecture. A TSL formula φ specifies a reactive system that in every time step processes a finite number of inputs $\mathbb{I}$ and produces a finite number of outputs $\mathbb{O}$. Furthermore, it uses cells $\mathbb{C}$ to store a value computed at time t, which can then be reused in the next time step $t+1$. An overview of the architecture of such a system is given in Fig. 4a. In terms of behavior, the environment produces infinite



(a) Architecture

Function Term:

$\tau_F := \mathtt{s_i} \mid \mathtt{f}\ \tau_F^0\ \tau_F^1\ \cdots\ \tau_F^{n-1}$

Predicate Term:

$\tau_P := \mathtt{p}\ \tau_F^0\ \tau_F^1\ \cdots\ \tau_F^{n-1}$

Update:

$[\![\mathtt{s_o} \leftarrowtail \tau_F]\!]$

(b) Term Definitions

Fig. 4. General architecture of reactive systems that are specified in TSL on the left, and the structure of function, predicate and updates on the right.

streams of input data, while the system uses pure (side-effect free) functions to transform the values of these input streams in every time step. After their transformation, the data values are either passed to an output stream or are passed to a cell, which pipes the output value from one time step back to the corresponding input value of the next. The behaviour of the system is captured by its infinite execution over time.

Function Terms, Predicate Terms, and Updates. In TSL we differentiate between two elements: we use purely functional transformations, reflected by functions $f \in \mathcal{F}$ and their compositions, and predicates $p \in \mathcal{P}$, used to control how data flows inside the system. To argue about both elements we use a term based notation, where we distinguish between function terms τ_F and predicate terms τ_P, respectively. Function terms are either constructed from inputs or cells ($\mathtt{s_i} \in \mathbb{I} \cup \mathbb{C}$), or from functions, recursively applied to a set of function terms. Predicate terms are constructed similarly, by applying a predicate to a set of function terms. Finally, an update takes the result of a function computation and passes it either to an output or a cell ($\mathtt{s_o} \in \mathbb{O} \cup \mathbb{C}$). An overview of the syntax of the different term notations is given in Fig. 4b. Note that we use curried argument notation similar to functional programming languages.

We denote sets of function and predicate terms, and updates by $\mathcal{T}_F$, $\mathcal{T}_P$ and $\mathcal{T}_U$, respectively, where $\mathcal{T}_P \subseteq \mathcal{T}_F$. We use $\mathbb{F}$ to denote the set of function literals and $\mathbb{P} \subseteq \mathbb{F}$ to denote the set of predicate literals, where the literals $\mathtt{s_i}$, $\mathtt{s_o}$, $\mathtt{f}$ and $\mathtt{p}$ are symbolic representations of inputs and cells, outputs and cells, functions, and predicates, respectively. Literals are used to construct terms as shown in Fig. 4b. Since we use a symbolic representation, functions and predicates are not tied to a specific implementation. However, we still classify them according to their arity, i.e., the number of function terms they are applied to, as well as by their type: input, output, cell, function or predicate. Furthermore, terms can be compared syntactically using the equivalence relation $\equiv$. To assign a semantic interpretation to functions, we use an assignment function $\langle\cdot\rangle\colon \mathbb{F} \to \mathcal{F}$.

Inputs, Outputs, and Computations. We consider momentary inputs $i \in \mathcal{V}^{[\mathbb{I}]}$, which are assignments of inputs $\mathtt{i} \in \mathbb{I}$ to values $v \in \mathcal{V}$. For the sake of readability let $\mathcal{I} = \mathcal{V}^{[\mathbb{I}]}$. Input streams are infinite sequences $\iota \in \mathcal{I}^\omega$ consisting of infinitely many momentary inputs.

Similarly, a momentary output $o \in \mathcal{V}^{[\mathbb{O}]}$ is an assignment of outputs $\mathtt{o} \in \mathbb{O}$ to values $v \in \mathcal{V}$, where we also use $\mathcal{O} = \mathcal{V}^{[\mathbb{O}]}$. Output streams are infinite sequences $\varrho \in \mathcal{O}^\omega$. To capture the behavior of a cell, we introduce the notion of a computation ς. A computation fixes the function terms that are used to compute outputs and cell updates, without fixing semantics of function literals. Intuitively, a computation only determines which function terms are used to compute an output, but abstracts from actually computing it.

The basic element of a computation is a computation step $c \in \mathcal{T}_F^{[\mathbb{O} \cup \mathbb{C}]}$, which is an assignment of outputs and cells $\mathtt{s_o} \in \mathbb{O} \cup \mathbb{C}$ to function terms $\tau_F \in \mathcal{T}_F$. For the sake of readability let $\mathcal{C} = \mathcal{T}_F^{[\mathbb{O} \cup \mathbb{C}]}$. A computation step fixes the control flow behaviour at a single point in time. A computation $\varsigma \in \mathcal{C}^\omega$ is an infinite sequence of computation steps.

As soon as input streams, and function and predicate implementations are known, computations can be turned into output streams. To this end, let $\langle\cdot\rangle \colon \mathbb{F} \to \mathcal{F}$ be some function assignment. Furthermore, assume that there are predefined constants $init_{\mathtt{c}} \in \mathcal{F} \cap \mathcal{V}$ for every cell $\mathtt{c} \in \mathbb{C}$, which provide an initial value for each stream at the initial point in time. To receive an output stream from a computation $\varsigma \in \mathcal{C}^\omega$ under the input stream ι, we use an evaluation function $\eta_{\langle\cdot\rangle} \colon \mathcal{C}^\omega \times \mathcal{I}^\omega \times \mathbb{N} \times \mathcal{T}_F \to \mathcal{V}$:

$$\eta_{\langle\cdot\rangle}(\varsigma, \iota, t, \mathtt{s_i}) = \begin{cases} \iota(t)(\mathtt{s_i}) & \text{if } \mathtt{s_i} \in \mathbb{I} \\ init_{\mathtt{s_i}} & \text{if } \mathtt{s_i} \in \mathbb{C} \ \wedge \ t = 0 \\ \eta_{\langle\cdot\rangle}(\varsigma, \iota, t-1, \varsigma(t-1)(\mathtt{s_i})) & \text{if } \mathtt{s_i} \in \mathbb{C} \ \wedge \ t > 0 \end{cases}$$

$$\eta_{\langle\cdot\rangle}(\varsigma, \iota, t, \mathtt{f}\ \tau_0\ \cdots\ \tau_{m-1}) = \langle \mathtt{f} \rangle\ \eta_{\langle\cdot\rangle}(\varsigma, \iota, t, \tau_0)\ \cdots\ \eta_{\langle\cdot\rangle}(\varsigma, \iota, t, \tau_{m-1})$$

Then $\varrho_{\langle\cdot\rangle,\varsigma,\iota} \in \mathcal{O}^\omega$ is defined via $\varrho_{\langle\cdot\rangle,\varsigma,\iota}(t)(\mathtt{o}) = \eta_{\langle\cdot\rangle}(\varsigma, \iota, t, \mathtt{o})$ for all $t \in \mathbb{N}$, $\mathtt{o} \in \mathbb{O}$.

Syntax. Every TSL formula φ is built according to the following grammar:

$$\varphi \ := \ \tau \in \mathcal{T}_P \cup \mathcal{T}_U \ \mid \ \neg\varphi \ \mid \ \varphi \wedge \varphi \ \mid \ \bigcirc\varphi \ \mid \ \varphi\, \mathcal{U}\, \varphi$$

An atomic proposition τ consists either of a predicate term, serving as a Boolean interface to the inputs, or of an update, enforcing a respective flow at the current point in time. Next, we have the Boolean operations via negation and conjunction, that allow us to express arbitrary Boolean combinations of predicate evaluations and updates. Finally, we have the temporal operator next: $\bigcirc\psi$, to specify the behavior at the next point in time and the temporal operator until: $\vartheta\, \mathcal{U}\, \psi$, which enforces a property ϑ to hold until the property ψ holds, where ψ must hold at some point in the future eventually.

Semantics. Formally, this leads to the following semantics. Let $\langle\cdot\rangle\colon \mathbb{F} \to \mathcal{F}$, $\iota \in \mathcal{I}^\omega$, and $\varsigma \in \mathcal{C}^\omega$ be given, then the validity of a TSL formula φ with respect to ς and ι is defined inductively over $t \in \mathbb{N}$ via:

$$
\begin{aligned}
\varsigma, \iota, t &\vDash_{\langle\cdot\rangle} \mathtt{p}\ \tau_0\ \cdots\ \tau_{m-1} &&:\Leftrightarrow\ \eta_{\langle\cdot\rangle}(\varsigma, \iota, t, \mathtt{p}\ \tau_0\ \cdots\ \tau_{m-1}) \\
\varsigma, \iota, t &\vDash_{\langle\cdot\rangle} [\![\mathtt{s} \leftarrowtail \tau]\!] &&:\Leftrightarrow\ \varsigma(t)(\mathtt{s}) \equiv \tau \\
\varsigma, \iota, t &\vDash_{\langle\cdot\rangle} \neg\psi &&:\Leftrightarrow\ \varsigma, \iota, t \nvDash_{\langle\cdot\rangle} \psi \\
\varsigma, \iota, t &\vDash_{\langle\cdot\rangle} \vartheta \wedge \psi &&:\Leftrightarrow\ \varsigma, \iota, t \vDash_{\langle\cdot\rangle} \vartheta\ \wedge\ \varsigma, \iota, t \vDash_{\langle\cdot\rangle} \psi \\
\varsigma, \iota, t &\vDash_{\langle\cdot\rangle} \bigcirc\psi &&:\Leftrightarrow\ \varsigma, \iota, t+1 \vDash_{\langle\cdot\rangle} \psi \\
\varsigma, \iota, t &\vDash_{\langle\cdot\rangle} \vartheta\,\mathcal{U}\,\psi &&:\Leftrightarrow\ \exists t'' \geq t.\ \forall t \leq t' < t''.\ \varsigma, \iota, t' \vDash_{\langle\cdot\rangle} \vartheta\ \wedge\ \varsigma, \iota, t'' \vDash_{\langle\cdot\rangle} \psi
\end{aligned}
$$

Consider that the satisfaction of a predicate depends on the current computation step and the steps of the past, while for updates it only depends on the current computation step. Furthermore, updates are only checked syntactically, while the satisfaction of predicates depends on the given assignment $\langle\cdot\rangle$ and the input stream ι. We say that ς and ι satisfy φ, denoted by $\varsigma, \iota \vDash_{\langle\cdot\rangle} \varphi$, if $\varsigma, \iota, 0 \vDash_{\langle\cdot\rangle} \varphi$.

Beside the basic operators, we have the standard derived Boolean operators, as well as the derived temporal operators: *release* $\varphi\,\mathcal{R}\,\psi \equiv \neg((\neg\psi)\,\mathcal{U}(\neg\varphi))$, *finally* $\diamondsuit\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$, *always* $\square\varphi \equiv \mathit{false}\,\mathcal{R}\,\varphi$, the *weak* version of *until* $\varphi\,\mathcal{W}\,\psi \equiv (\varphi\,\mathcal{U}\,\psi) \vee (\square\varphi)$, and *as soon as* $\varphi\,\mathcal{A}\,\psi \equiv \neg\psi\,\mathcal{W}(\psi \wedge \varphi)$.

Realizability. We are interested in the following realizability problem: given a TSL formula φ, is there a strategy $\sigma \in \mathcal{C}^{[\mathcal{I}^+]}$ such that for every input $\iota \in \mathcal{I}^\omega$ and function implementation $\langle\cdot\rangle\colon \mathbb{F} \to \mathcal{F}$, the branch $\sigma \wr \iota$ satisfies φ, i.e.,

$$\exists \sigma \in \mathcal{C}^{[\mathcal{I}^+]}.\ \forall \iota \in \mathcal{I}^\omega.\ \forall \langle\cdot\rangle\colon \mathbb{F} \to \mathcal{F}.\ \sigma \wr \iota, \iota \vDash_{\langle\cdot\rangle} \varphi$$

If such a strategy σ exists, we say σ realizes φ. If we additionally ask for a concrete instantiation of σ, we consider the synthesis problem of TSL.

5 TSL Properties

In order to synthesize programs from TSL specifications, we give an overview of the first part of our synthesis process, as shown in Fig. 1. First we show how to approximate the semantics of TSL through a reduction to LTL. However, due to the approximation, finding a realizable strategy immediately may fail. Our solution is a CEGAR loop that improves the approximation. This CEGAR loop is necessary, because the realizability problem of TSL is undecidable in general.

Approximating TSL with LTL. We approximate TSL formulas with weaker LTL formulas. The approximation reinterprets the syntactic elements, $\mathcal{T}_P$ and $\mathcal{T}_U$, as atomic propositions for LTL. This strips away the semantic meaning of the function application and assignment in TSL, which we reconstruct by later adding assumptions lazily to the LTL formula.

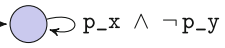
Formally, let $\mathcal{T}_P$ and $\mathcal{T}_U$ be the finite sets of predicate terms and updates, which appear in φ_{TSL}, respectively. For every assigned signal, we partition $\mathcal{T}_U$ into $\biguplus_{\mathtt{s}_\circ \in \mathbb{O} \cup \mathbb{C}} \mathcal{T}_U^{\mathtt{s}_\circ}$. For every $\mathtt{c} \in \mathbb{C}$ let $\mathcal{T}_{U/\mathtt{id}}^{\mathtt{c}} = \mathcal{T}_U^{\mathtt{c}} \cup \{[\![\mathtt{c} \leftarrowtail \mathtt{c}]\!]\}$, for $\mathtt{o} \in \mathbb{O}$ let

$$\square\,([\![\mathtt{y} \leftarrow \mathtt{y}]\!] \vee [\![\mathtt{y} \leftarrow \mathtt{x}]\!]) \wedge \diamondsuit\, \mathtt{p\ x} \rightarrow \diamondsuit\, \mathtt{p\ y}$$

(a) TSL specification

$$\square\, \neg(\mathtt{y\_to\_y} \wedge \mathtt{x\_to\_y}) \wedge \square\,(\mathtt{y\_to\_y} \vee \mathtt{x\_to\_y}) \wedge \diamondsuit\, \mathtt{p\_x} \rightarrow \diamondsuit\, \mathtt{p\_y}$$

(b) initial approximation



(c) spurious counter-strategy

Fig. 5. A TSL specification (a) with input `x` and cell `y` that is realizable. A winning strategy is to save `x` to `y` as soon as `p(x)` is satisfied. However, the initial approximation (b), that is passed to an LTL synthesis solver, is unrealizable, as proven through the counter-strategy (c) returned by the LTL solver.

$\mathcal{T}^{\mathtt{o}}_{U/\mathtt{id}} = \mathcal{T}^{\mathtt{o}}_{U}$, and let $\mathcal{T}_{U/\mathtt{id}} = \bigcup_{\mathtt{s_o} \in \mathbb{O} \cup \mathbb{C}} \mathcal{T}^{\mathtt{s_o}}_{U/\mathtt{id}}$. We construct the LTL formula φ_{LTL} over the input propositions $\mathcal{T}_P$ and output propositions $\mathcal{T}_{U/\mathtt{id}}$ as follows:

$$\varphi_{LTL} = \square\Big(\bigwedge_{\mathtt{s_o} \in \mathbb{O} \cup \mathbb{C}} \bigvee_{\tau \in \mathcal{T}^{\mathtt{s_o}}_{U/\mathtt{id}}} \Big(\tau \wedge \bigwedge_{\tau' \in \mathcal{T}^{\mathtt{s_o}}_{U/\mathtt{id}} \setminus \{\tau\}} \neg \tau'\Big)\Big) \wedge \textsc{SyntacticConversion}(\varphi_{TSL})$$

Intuitively, the first part of the equation partially reconstructs the semantic meaning of updates by ensuring that a signal is not updated with multiple values at a time. The second part extracts the reactive constraints of the TSL formula without the semantic meaning of functions and updates.

Theorem 1 ([19]). *If φ_{LTL} is realizable, then φ_{TSL} is realizable.*

Note that unrealizability of φ_{LTL} does not imply that φ_{TSL} is unrealizable. It may be that we have not added sufficiently many environment assumptions to the approximation in order for the system to produce a realizing strategy.

Example. As an example, we present a simple TSL specification in Fig. 5a. The specification asserts that the environment provides an input `x` for which the predicate `p x` will be satisfied eventually. The system must guarantee that eventually `p y` holds. According to the semantics of TSL the formula is realizable. The system can take the value of `x` when `p x` is true and save it to `y`, thus guaranteeing that `p y` is satisfied eventually. This is in contrast to LTL, which has no semantics for pure functions - taking the evaluation of `p y` as an environmentally controlled value that does not need to obey the consistency of a pure function.

Refining the LTL Approximation. It is possible that the LTL solver returns a counter-strategy for the environment although the original TSL specification is realizable. We call such a counter-strategy *spurious* as it exploits the additional freedom of LTL to violate the purity of predicates as made possible by the underapproximation. Formally, a counter-strategy is an infinite tree $\pi\colon \mathcal{C}^* \rightarrow 2^{\mathcal{T}_P}$, which provides predicate evaluations in response to possible update assignments of function terms $\tau_F \in \mathcal{T}_F$ to outputs $\mathtt{o} \in \mathbb{O}$. W.l.o.g. we can assume that $\mathbb{O}$, $\mathcal{T}_F$ and $\mathcal{T}_P$ are finite, as they can always be restricted to the outputs and terms that appear in the formula. A counter-strategy is spurious, iff there is a branch $\pi \wr \varsigma$ for some computation $\varsigma \in \mathcal{C}^\omega$, for which the strategy chooses an inconsistent evaluation of two equal predicate terms at different points in time, i.e.,

Algorithm 1. Check-Spuriousness

Input: bound b, counter-strategy $\pi\colon \mathcal{C}^* \to 2^{\mathcal{T}_P}$ (finitely represented using m states)

1: **for all** $v \in \mathcal{C}^{m \cdot b}$, $\tau_P \in \mathcal{T}_P$, $t, t' \in \{0, 1, \ldots, m \cdot b - 1\}$ **do**
2: **if** $\eta_{\langle\cdot\rangle_{\mathtt{id}}}(v, \iota_{\mathtt{id}}, t, \tau_P) \equiv \eta_{\langle\cdot\rangle_{\mathtt{id}}}(v, \iota_{\mathtt{id}}, t', \tau_P) \wedge$
$\tau_P \in \pi(v_0 \ldots v_{t-1}) \wedge \tau_P \notin \pi(v_0 \ldots v_{t'-1})$ **then**
3: $w \leftarrow \mathtt{reduce}\,(v, \tau_P, t, t')$
4: **return** $\square\big(\bigwedge_{i=0}^{t-1} \bigcirc^i w_i \wedge \bigwedge_{i=0}^{t'-1} \bigcirc^i w_i \rightarrow (\bigcirc^t \tau_P \leftrightarrow \bigcirc^{t'} \tau_P)\big)$
5: **return** ‘‘non-spurious’’

$$\exists \varsigma \in \mathcal{C}^\omega.\ \exists t, t' \in \mathbb{N}.\ \exists \tau_P \in \mathcal{T}_P.$$
$$\tau_P \in \pi(\varsigma(0)\varsigma(1)\ldots\varsigma(t-1)) \wedge \tau_P \notin \pi(\varsigma(0)\varsigma(1)\ldots\varsigma(t'-1)) \wedge$$
$$\forall \langle\cdot\rangle\colon \mathbb{F} \rightarrow \mathcal{F}.\ \eta_{\langle\cdot\rangle}(\varsigma, \pi\wr\varsigma, t, \tau_P) = \eta_{\langle\cdot\rangle}(\varsigma, \pi\wr\varsigma, t', \tau_P).$$

Note that a non-spurious strategy can be inconsistent along multiple branches. Due to the definition of realizability the environment can choose function and predicate assignments differently against every system strategy accordingly.

By purity of predicates in TSL the environment is forced to always return the same value for predicate evaluations on equal values. However, this semantic property cannot be enforced implicitly in LTL. To resolve this issue we use the returned counter-strategy to identify spurious behavior in order to strengthen the LTL underapproximation with additional environment assumptions. After adding the derived assumptions, we re-execute the LTL synthesizer to check whether the added assumptions are sufficient in order to obtain a winning strategy for the system. If the solver still returns a spurious strategy, we continue the loop in a CEGAR fashion until the set of added assumptions is sufficiently complete. However, if a non-spurious strategy is returned, we have found a proof that the given TSL specification is indeed unrealizable and terminate.

Algorithm 1 shows how a returned counter-strategy π is checked for being spurious. To this end, it is sufficient to check π against system strategies bounded by the given bound b, as we use bounded synthesis [20]. Furthermore, we can assume w.l.o.g. that π is given by a finite state representation, which is always possible due to the finite model guarantees of LTL. Also note that π, as it is returned by the LTL synthesizer, responds to sequences of sets of updates $(2^{\mathcal{T}_{U/\mathtt{id}}})^*$. However, in our case $(2^{\mathcal{T}_{U/\mathtt{id}}})^*$ is an alternative representation of $\mathcal{C}^*$, due to the additional “single update” constraints added during the construction of φ_{LTL}.

The algorithm iterates over all possible responses $v \in \mathcal{C}^{m \cdot b}$ of the system up to depth $m \cdot b$. This is sufficient, since any deeper exploration would result in a state repetition of the cross-product of the finite state representation of π and any system strategy bounded by b. Hence, the same behaviour could also be generated by a sequence smaller than $m \cdot b$. At the same time, the algorithm iterates over predicates $\tau_P \in \mathcal{T}_P$ appearing in φ_{TSL} and times t and t' smaller than $m \cdot b$. For each of these elements, spuriousness is checked by comparing the output of π for the evaluation of τ_P at times t and t', which should only differ if the inputs to the predicates are different as well. This can only happen, if the

passed input terms have been constructed differently over the past. We check it by using the evaluation function η equipped with the identity assignment $\langle\cdot\rangle_{\mathtt{id}}\colon \mathbb{F} \to \mathbb{F}$, with $\langle \mathtt{f} \rangle_{\mathtt{id}} = \mathtt{f}$ for all $\mathtt{f} \in \mathbb{F}$, and the input sequence $\iota_{\mathtt{id}}$, with $\iota_{\mathtt{id}}(t)(\mathtt{i}) = (t, \mathtt{i})$ for all $t \in \mathbb{N}$ and $\mathtt{i} \in \mathbb{I}$, that always generates a fresh input. Syntactic inequality of $\eta_{\langle\cdot\rangle_{\mathtt{id}}}(v, \iota_{\mathtt{id}}, t, \tau_P)$ and $\eta_{\langle\cdot\rangle_{\mathtt{id}}}(v, \iota_{\mathtt{id}}, t', \tau_P)$ then is a sufficient condition for the existence of an assignment $\langle\cdot\rangle\colon \mathbb{F} \to \mathcal{F}$, for which τ_P evaluates differently at times t and t'.

If spurious behaviour of π could be found, then the revealing response $v \in \mathcal{C}^*$ is first simplified using `reduce`, which reduces v again to a sequence of sets of updates $w \in (2^{\mathcal{T}_{U/\mathtt{id}}})^*$ and removes updates that do not affect the behavior of τ_P at the times t and t' to accelerate the termination of the CEGAR loop. Afterwards, the sequence w is turned into a new assumption that prohibits the spurious behavior, generalized to prevent it even at arbitrary points in time.

As an example of this process, reconsider the spurious counter-strategy of Fig. 5c. Already after the first system response $[\![\mathtt{y} \leftarrowtail \mathtt{x}]\!]$, the environment produces an inconsistency by evaluating $\mathtt{p}\ \mathtt{x}$ and $\mathtt{p}\ \mathtt{y}$ differently. This is inconsistent, as the cell $\mathtt{y}$ holds the same value at time $t = 1$ as the input $\mathtt{x}$ at time $t = 0$. Using Algorithm 1 we generate the new assumption $\square([\![\mathtt{y} \leftarrowtail \mathtt{x}]\!] \rightarrow (\mathtt{p}\ \mathtt{x} \leftrightarrow \bigcirc \mathtt{p}\ \mathtt{y}))$. After adding this strengthening the LTL synthesizer returns a realizability result.

Undecidability. Although we can approximate the semantics of TSL with LTL, there are TSL formulas that cannot be expressed as LTL formulas of finite size.

Theorem 2 ([19]). *The realizability problem of TSL is undecidable.*

6 TSL Synthesis

Our synthesis framework provides a modular refinement process to synthesize executables from TSL specifications, as depicted in Fig. 1. The user initially provides a TSL specification over predicate and function terms. At the end of the procedure, the user receives an executable to control a reactive system.

The first step of our method answers the synthesis question of TSL: if the specification is realizable, then a control flow model is returned. To this end, an intermediate translation to LTL is used, utilizing an LTL synthesis solver that produces circuits in the AIGER format. If the specification is realizable, the resulting control flow model is turned into Haskell code, which is implemented as an independent Haskell module. The user has the choice between two different targets: a module built on Arrows, which is compatible with any Arrowized FRP library, or a module built on Applicative, which supports Applicative FRP libraries. Our procedure generates a single Haskell module per TSL specification. This makes naturally decomposing a project according to individual tasks possible. Each module provides a single component, which is parameterized by their initial state and the pure function and predicate transformations. As soon as these are provided as part of the surrounding project context, a final executable can be generated by compiling the Haskell code.

An important feature of our synthesis approach is that implementations for the terms used in the specification are only required after synthesis. This allows

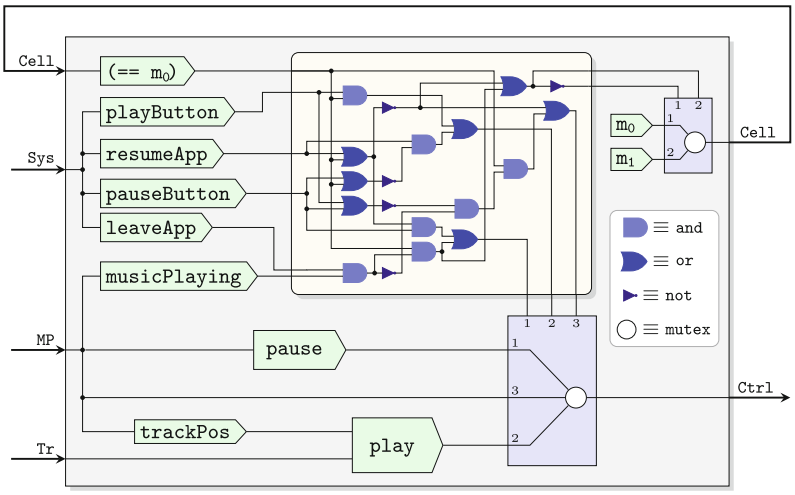


Fig. 6. Example CFM of the music player generated from a TSL specification.

the user to explore several possible specifications before deciding on any term implementations.

Control Flow Model. The first step of our approach is the synthesis of a *Control Flow Model* $\mathcal{M}$ (CFM) from the given TSL specification φ, which provides us with a uniform representation of the control flow structure of our final program.

Formally, a CFM $\mathcal{M}$ is a tuple $\mathcal{M} = (\mathbb{I}, \mathbb{O}, \mathbb{C}, V, \ell, \delta)$, where $\mathbb{I}$ is a finite set of inputs, $\mathbb{O}$ is a finite set of outputs, $\mathbb{C}$ is a finite set of cells, V is a finite set of vertices, $\ell \colon V \to \mathbb{F}$ assigns a vertex a function $\mathtt{f} \in \mathbb{F}$ or a predicate $\mathtt{p} \in \mathbb{P}$, and

$$\delta \colon (\mathbb{O} \cup \mathbb{C} \cup V) \times \mathbb{N} \to (\mathbb{I} \cup \mathbb{C} \cup V \cup \{\bot\})$$

is a dependency relation that relates every output, cell, and vertex of the CFM with $n \in \mathbb{N}$ arguments, which are either inputs, cells, or vertices. Outputs and cells $\mathtt{s} \in \mathbb{O} \cup \mathbb{C}$ always have only a single argument, i.e., $\delta(\mathtt{s}, 0) \not\equiv \bot$ and $\forall m > 0.$ $\delta(\mathtt{s}, m) \equiv \bot$, while for vertices $x \in V$ the number of arguments $n \in \mathbb{N}$ align with the arity of the assigned function or predicate $\ell(x)$, i.e., $\forall m \in \mathbb{N}.\ \delta(x, m) \equiv \bot \leftrightarrow m > n$. A CFM is valid if it does not contain circular dependencies, i.e., on every cycle induced by δ there must lie at least a single cell. We only consider valid CFMs.

An example CFM for our music player of Sect. 2 is depicted in Fig. 6. Inputs $\mathbb{I}$ come from the left and outputs $\mathbb{O}$ leave on the right. The example contains a single cell $\mathtt{c} \in \mathbb{C}$, which holds the stateful memory `Cell`, introduced during synthesis for the module. The green, arrow shaped boxes depict vertices V, which are labeled with functions and predicates names, according to ℓ. For the Boolean decisions that define δ, we use circuit symbols for conjunction, disjunction, and negation. Boolean decisions are piped to a multiplexer gate that selects the respective update streams. This allows each update stream to be passed to an

output stream if and only if the respective Boolean trigger evaluates positively, while our construction ensures mutual exclusion on the Boolean triggers. For code generation, the logic gates are implemented using the corresponding dedicated Boolean functions. After building a control structure, we assign semantics to functions and predicates by providing implementations. To this end, we use Functional Reactive Programming (FRP). Prior work has established Causal Commutative Arrows (CCA) as an FRP language pattern equivalent to a CFM [33,34,53]. CCAs are an abstraction subsumed by other functional reactive programming abstractions, such as Monads, Applicative and Arrows [32,33]. There are many FRP libraries using Monads [11,14,42], Applicative [2,3,23,48], or Arrows [10,39,41,51], and since every Monad is also an Applicative and Applicative/Arrows both are universal design patterns, we can give uniform translations to all of these libraries using translations to just Applicative and Arrows. Both translations are possible due to the flexible notion of a CFM.

In the last step, the synthesized FRP program is compiled into an executable, using the provided function and predicate implementations. This step is not fixed to a single compiler implementation, but in fact can use any FRP compiler (or library) that supports a language abstraction at least as expressive as CCA. For example, instead of creating an Android music player app, we could target an FRP web interface [48] to create an online music player, or an embedded FRP library [23] to instantiate the player on a computationally more restricted device. By using the strong core of CCA, we even can directly implement the player in hardware, which is for example possible with the CλaSH compiler [3]. Note that we still need separate implementations for functions and predicates for each target. However, the specification and synthesized CFM always stay the same.

7 Experimental Results

To evaluate our synthesis procedure we implemented a tool that follows the structure of Fig. 1. It first encodes the given TSL specification in LTL and then refines it until an LTL solver either produces a realizability result or returns a non-spurious counter-strategy. For LTL synthesis we use the bounded synthesis tool BoSy [15]. As soon as we get a realizing strategy it is translated to a corresponding CFM. Then, we generate the FRP program structure. Finally, after providing function implementations the result is compiled into an executable.

To demonstrate the effectiveness of synthesizing TSL, we applied our tool to a collection of benchmarks from different application domains, listed in Table 1. Every benchmark class consists of multiple specifications, addressing different features of TSL. We created all specifications from scratch, where we took care that they either relate to existing textual specifications, or real world scenarios. A short description of each benchmark class is given in [19].

For every benchmark, we report the synthesis time and the size of the synthesized CFM, split into the number of cells ($|\mathbb{C}_{\mathcal{M}}|$) and vertices ($|V_{\mathcal{M}}|$) used. The synthesized CFM may use more cells than the original TSL specification if synthesis requires more memory in order to realize a correct control flow.

Table 1. Number of cells $|\mathbb{C}_{\mathcal{M}}|$ and vertices $|V_{\mathcal{M}}|$ of the resulting CFM $\mathcal{M}$ and synthesis times for a collection of TSL specifications φ. A * indicates that the benchmark additionally has an initial condition as part of the specification.

Benchmark (φ)	$\lvert\varphi\rvert$	$\lvert\mathbb{I}\rvert$	$\lvert\mathbb{O}\rvert$	$\lvert\mathbb{P}\rvert$	$\lvert\mathbb{F}\rvert$	$\lvert\mathbb{C}_{\mathcal{M}}\rvert$	$\lvert V_{\mathcal{M}}\rvert$	Synthesis Time (s)
Button								
default	7	1	2	1	3	3	8	0.364
Music App								
simple	91	3	1	4	7	2	25	0.77
system feedback	103	3	1	5	8	2	31	0.572
motivating example	87	3	1	5	8	2	70	1.783
FRPZoo								
scenario_0	54	1	3	2	8	4	36	1.876
scenario_5	50	1	3	2	7	4	32	1.196
scenario_{10}	48	1	3	2	7	4	32	1.161
Escalator								
non-reactive	8	0	1	0	1	2	4	0.370
non-counting	15	2	1	2	4	2	19	0.304
counting	34	2	2	3	7	3	23	0.527
counting*	43	2	2	3	8	4	43	0.621
bidirectional	111	2	2	5	10	3	214	4.555
bidirectional*	124	2	2	5	11	4	287	16.213
smart	45	2	1	2	4	4	159	24.016
Slider								
default	50	1	1	2	4	2	15	0.664
scored	67	1	3	4	8	4	62	3.965
delayed	71	1	3	4	8	5	159	7.194
Haskell-TORCS								
simple	40	5	3	2	16	4	37	0.680
advanced								
gearing	23	4	1	1	3	2	7	0.403
accelerating	15	2	2	2	6	3	11	0.391
steering								
simple	45	2	1	4	6	2	31	0.459
improved	100	2	2	4	10	3	26	1.347
smart	76	3	2	4	8	5	227	3.375

Table 2. Set of programs that use purity to keep one or two counters in range. Synthesis needs multiple refinements of the specification to proof realizability.

Benchmark (φ)	$\lvert\varphi\rvert$	$\lvert\mathbb{I}\rvert$	$\lvert\mathbb{O}\rvert$	$\lvert\mathbb{P}\rvert$	$\lvert\mathbb{F}\rvert$	$\lvert\mathbb{C}_{\mathcal{M}}\rvert$	$\lvert V_{\mathcal{M}}\rvert$	Refinements	Synthesis Time (s)
inrange-single	23	2	1	2	4	2	21	3	0.690
inrange-two	51	3	3	4	7	4	440	6	173.132
graphical-single	55	2	3	2	6	4	343	9	1767.948
graphical-two	113	3	5	4	9	-	-	-	¿ 10000

The synthesis was executed on a quad-core Intel Xeon processor (E3-1271 v3, 3.6GHz, 32 GB RAM, PC1600, ECC), running Ubuntu 64bit LTS 16.04.

The experiments of Table 1 show that TSL successfully lifts the applicability of synthesis from the Boolean domain to arbitrary data domains, allowing for new applications that utilize every level of required abstraction. For all benchmarks we always found a realizable system within a reasonable amount of time, where the results often required synthesized cells to realize the control flow behavior.

We also considered a preliminary set of benchmarks that require multiple refinement steps to be synthesizable. An overview of the results is given in Table 2. The benchmarks are inspired by examples of the Reactive Banana FRP library [2]. Here, purity of function and predicate applications must be utilized by the system to ensure that the value of one or two counters never goes out of range. Thereby, the system not only needs purity to verify this condition, but also to take the correct decisions in the resulting implementation to be synthesized.

8 Related Work

Our approach builds on the rich body of work on reactive synthesis, see [17] for a survey. The classic reactive synthesis problem is the construction of a finite-state machine that satisfies a specification in a temporal logic like LTL. Our approach differs from the classic problem in its connection to an actual programming paradigm, namely FRP, and its separation of control and data.

The synthesis of *reactive programs*, rather than finite-state machines, has previously been studied for standard temporal logic [21,35]. Because there is no separation of control and data, these approaches do not directly scale to realistic applications. With regard to FRP, a *Curry-Howard correspondence* between LTL and FRP in a dependently typed language was discovered [28,29] and used to prove properties of FRP programs [8,30]. However, our paper is the first, to the best of our knowledge, to study the synthesis of FRP programs from temporal specifications.

The idea to separate control and data has appeared, on a smaller scale, in the synthesis with *identifiers*, where identifiers, such as the number of a client in a mutual exclusion protocol, are treated symbolically [13]. *Uninterpreted functions* have been used to abstract data-related computational details in the synthesis of synchronization primitives for complex programs [5]. Another connection to other synthesis approaches is our CEGAR loop. Similar *refinement loops* also appear in other synthesis appraches, however with a different purpose, such as the refinement of environment assumptions [1].

So far, there is no immediate connection between our approach and the substantial work on *deductive* and *inductive synthesis*, which is specifically concerned with the data-transformation aspects of programs [16,31,40,47,49,50]. Typically, these approaches are focussed on non-reactive sequential programs. An integration of deductive and inductive techniques into our approach for reactive systems is a very promising direction for future work. Abstraction-based synthesis [4,12,24,37] may potentially provide a link between the approaches.

9 Conclusions

We have introduced Temporal Stream Logic, which allows the user to specify the control flow of a reactive program. The logic cleanly separates control from complex data, forming the foundation for our procedure to synthesize FRP programs. By utilizing the purity of function transformations our logic scales independently of the complexity of the data to be handled. While we have shown that scalability comes at the cost of undecidability, we addressed this issue by using a CEGAR loop, which lazily refines the underapproximation until either a realizing system implementation or an unrealizability proof is found.

Our experiments indicate that TSL synthesis works well in practice and on a wide range of programming applications. TSL also provides the foundations for further extensions. For example, a user may want to fix the semantics for a subset of the functions and predicates. Such refinements can be implemented as part of a much richer *TSL Modulo Theory* framework.

References

1. Alur, R., Moarref, S., Topcu, U.: Counter-strategy guided refinement of GR(1) temporal logic specifications. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, 20–23 October 2013, pp. 26–33. IEEE (2013). http://ieeexplore.ieee.org/document/6679387/
2. Apfelmus, H.: Reactive-banana. Haskell library (2012). http://www.haskell.org/haskellwiki/Reactive-banana
3. Baaij, C.: Digital circuit in CλaSH: functional specifications and type-directed synthesis. Ph.D. thesis, University of Twente, January 2015. https://doi.org/10.3990/1.9789036538039, eemcs-eprint-23939
4. Beyene, T.A., Chaudhuri, S., Popeea, C., Rybalchenko, A.: A constraint-based approach to solving games on infinite graphs. In: Jagannathan and Sewell [26], pp. 221–234. https://doi.org/10.1145/2535838.2535860, http://doi.acm.org/10.1145/2535838.2535860
5. Bloem, R., Hofferek, G., Könighofer, B., Könighofer, R., Ausserlechner, S., Spork, R.: Synthesis of synchronization using uninterpreted functions. In: Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, 21–24 October 2014, pp. 35–42. IEEE (2014). https://doi.org/10.1109/FMCAD.2014.6987593
6. Bloem, R., Jacobs, S., Khalimov, A.: Parameterized synthesis case study: AMBA AHB. In: Chatterjee, K., Ehlers, R., Jha, S. (eds.) Proceedings 3rd Workshop on Synthesis, SYNT 2014. EPTCS, Vienna, Austria, 23–24 July 2014, vol. 157, pp. 68–83 (2014). https://doi.org/10.4204/EPTCS.157.9
7. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. J. Comput. Syst. Sci. **78**(3), 911–938 (2012). https://doi.org/10.1016/j.jcss.2011.08.007
8. Cave, A., Ferreira, F., Panangaden, P., Pientka, B.: Fair reactive programming. In: Jagannathan and Sewell [26], pp. 361–372. https://doi.org/10.1145/2535838.2535881, http://doi.acm.org/10.1145/2535838.2535881
9. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643

10. Courtney, A., Nilsson, H., Peterson, J.: The yampa arcade. In: Proceedings of the ACM SIGPLAN Workshop on Haskell, Haskell 2003, Uppsala, Sweden, 28 August 2003, pp. 7–18. ACM, (2003). https://doi.org/10.1145/871895.871897, http://doi.acm.org/10.1145/871895.871897
11. Czaplicki, E., Chong, S.: Asynchronous functional reactive programming for Guis. In: Boehm, H., Flanagan, C. (eds.) ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2013, Seattle, WA, USA, 16–19 June 2013, pp. 411–422. ACM (2013). https://dl.acm.org/citation.cfm?doid=2462156.2462161, http://doi.acm.org/10.1145/2462156.2462161
12. Dimitrova, R., Finkbeiner, B.: Counterexample-guided synthesis of observation predicates. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 107–122. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33365-1_9
13. Ehlers, R., Seshia, S.A., Kress-Gazit, H.: Synthesis with identifiers. In: McMillan, K.L., Rival, X. (eds.) VMCAI 2014. LNCS, vol. 8318, pp. 415–433. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54013-4_23
14. Elliott, C., Hudak, P.: Functional reactive animation. In: Jones, S.L.P., Tofte, M., Berman, A.M. (eds.) Proceedings of the 1997 ACM SIGPLAN International Conference on Functional Programming (ICFP 1997), Amsterdam, The Netherlands, 9–11 June 1997, pp. 263–273. ACM (1997). https://doi.org/10.1145/258948.258973, http://doi.acm.org/10.1145/258948.258973
15. Faymonville, P., Finkbeiner, B., Tentrup, L.: BoSy: an experimentation framework for bounded synthesis. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 325–332. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_17
16. Feser, J.K., Chaudhuri, S., Dillig, I.: Synthesizing data structure transformations from input-output examples. In: Grove and Blackburn [22], pp. 229–239. https://doi.org/10.1145/2737924.2737977, http://doi.acm.org/10.1145/2737924.2737977
17. Finkbeiner, B.: Synthesis of reactive systems. In: Esparza, J., Grumberg, O., Sickert, S. (eds.) Dependable Software Systems Engineering. NATO Science for Peace and Security Series, D: Information and Communication Security, vol. 45, pp. 72–98. IOS Press (2016)
18. Finkbeiner, B., Klein, F., Piskac, R., Santolucito, M.: Vehicle platooning simulations with functional reactive programming. In: Proceedings of the 1st International Workshop on Safe Control of Connected and Autonomous Vehicles, SCAV@CPSWeek 2017, Pittsburgh, PA, USA, 21 April 2017, pp. 43–47. ACM, (2017). https://doi.org/10.1145/3055378.3055385, http://doi.acm.org/10.1145/3055378.3055385
19. Finkbeiner, B., Klein, F., Piskac, R., Santolucito, M.: Temporal stream logic: Synthesis beyond the bools. CoRR abs/1712.00246 (2019). http://arxiv.org/abs/1712.00246
20. Finkbeiner, B., Schewe, S.: Bounded synthesis. STTT **15**(5–6), 519–539 (2013). https://doi.org/10.1007/s10009-012-0228-z
21. Gerstacker, C., Klein, F., Finkbeiner, B.: Bounded synthesis of reactive programs. In: Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, 7–10 October 2018, Proceedings, pp. 441–457 (2018). https://doi.org/10.1007/978-3-030-01090-4_26
22. Grove, D., Blackburn, S. (eds.): Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, 15–17 June 2015. ACM (2015). http://dl.acm.org/citation.cfm?id=2737924

23. Helbling, C., Guyer, S.Z.: Juniper: a functional reactive programming language for the arduino. In: Janin and Sperber [27], pp. 8–16. https://doi.org/10.1145/2975980.2975982, http://doi.acm.org/10.1145/2975980.2975982
24. Hsu, K., Majumdar, R., Mallik, K., Schmuck, A.K.: Multi-layered abstraction-based controller synthesis for continuous-time systems. In: Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), pp. 120–129. ACM (2018)
25. Jacobs, S., et al.: The 4th reactive synthesis competition (SYNTCOMP 2017): Benchmarks, participants and results. In: SYNT 2017. EPTCS, vol. 260, pp. 116–143 (2017). https://doi.org/10.4204/EPTCS.260.10
26. Jagannathan, S., Sewell, P. (eds.): The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2014, San Diego, CA, USA, 20–21 January 2014. ACM (2014). http://dl.acm.org/citation.cfm?id=2535838
27. Janin, D., Sperber, M. (eds.): Proceedings of the 4th International Workshop on Functional Art, Music, Modelling, and Design, FARM@ICFP 2016, Nara, Japan, 24 September 2016. ACM (2016). https://doi.org/10.1145/2975980, http://doi.acm.org/10.1145/2975980
28. Jeffrey, A.: LTL types FRP: linear-time temporal logic propositions as types, proofs as functional reactive programs. In: Claessen, K., Swamy, N. (eds.) Proceedings of the sixth workshop on Programming Languages meets Program Verification, PLPV 2012, Philadelphia, PA, USA, 24 January 2012, pp. 49–60. ACM (2012). https://doi.org/10.1145/2103776.2103783, http://doi.acm.org/10.1145/2103776.2103783
29. Jeltsch, W.: Towards a common categorical semantics for linear-time temporal logic and functional reactive programming. Electr. Notes Theor. Comput. Sci. **286**, 229–242 (2012). https://doi.org/10.1016/j.entcs.2012.08.015
30. Krishnaswami, N.R.: Higher-order functional reactive programming without space-time leaks. In: Morrisett, G., Uustalu, T. (eds.) ACM SIGPLAN International Conference on Functional Programming, ICFP 2013, Boston, MA, USA, 25–27 September 2013, pp. 221–232. ACM (2013). https://doi.org/10.1145/2500365.2500588, http://doi.acm.org/10.1145/2500365.2500588
31. Kuncak, V., Mayer, M., Piskac, R., Suter, P.: Comfusy: a tool for complete functional synthesis. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 430–433. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_38
32. Lindley, S., Wadler, P., Yallop, J.: Idioms are oblivious, arrows are meticulous, monads are promiscuous. Electr. Notes Theor. Comput. Sci. **229**(5), 97–117 (2011). https://doi.org/10.1016/j.entcs.2011.02.018
33. Liu, H., Cheng, E., Hudak, P.: Causal commutative arrows. J. Funct. Program. **21**(4–5), 467–496 (2011). https://doi.org/10.1017/S0956796811000153
34. Liu, H., Hudak, P.: Plugging a space leak with an arrow. Electr. Notes Theor. Comput. Sci. **193**, 29–45 (2007). https://doi.org/10.1016/j.entcs.2007.10.006
35. Madhusudan, P.: Synthesizing reactive programs. In: Bezem, M. (ed.) Computer Science Logic, 25th International Workshop/20th Annual Conference of the EACSL, CSL 2011, Bergen, Norway, 12–15 September 2011, Proceedings. LIPIcs, vol. 12, pp. 428–442. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011). https://doi.org/10.4230/LIPIcs.CSL.2011.428
36. Mainland, G. (ed.): Proceedings of the 9th International Symposium on Haskell, Haskell 2016, Nara, Japan, 22–23 September 2016. ACM (2016). https://doi.org/10.1145/2976002, http://doi.acm.org/10.1145/2976002

37. Mallik, K., Schmuck, A.K., Soudjani, S., Majumdar, R.: Compositional abstraction-based controller synthesis for continuous-time systems. arXiv preprint arXiv:1612.08515 (2016)
38. Manna, Z., Waldinger, R.: A deductive approach to program synthesis. ACM Trans. Program. Lang. Syst. **2**(1), 90–121 (1980). https://doi.org/10.1145/357084.357090
39. Murphy, T.E.: A livecoding semantics for functional reactive programming. In: Janin and Sperber [27], pp. 48–53. https://doi.org/10.1145/2975980.2975986http://doi.acm.org/10.1145/2975980.2975986
40. Osera, P., Zdancewic, S.: Type-and-example-directed program synthesis. In: Grove and Blackburn [22], pp. 619–630. https://doi.org/10.1145/2737924.2738007, http://doi.acm.org/10.1145/2737924.2738007
41. Perez, I., Bärenz, M., Nilsson, H.: Functional reactive programming, refactored. In: Mainland [36], pp. 33–44. https://doi.org/10.1145/2976002.2976010, http://doi.acm.org/10.1145/2976002.2976010
42. van der Ploeg, A., Claessen, K.: Practical principled FRP: forget the past, change the future, FRPNow! In: Fisher, K., Reppy, J.H. (eds.) Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, 1–3 September 2015, pp. 302–314. ACM (2015). https://doi.org/10.1145/2784731.2784752, http://doi.acm.org/10.1145/2784731.2784752
43. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October–1 November 1977, pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32
44. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: Ausiello, G., Dezani-Ciancaglini, M., Della Rocca, S.R. (eds.) ICALP 1989. LNCS, vol. 372, pp. 652–671. Springer, Heidelberg (1989). https://doi.org/10.1007/BFb0035790
45. Post, E.L.: A variant of a recursively unsolvable problem. Bull. Am. Math. Soc. **52**(4), 264–268 (1946). http://projecteuclid.org/euclid.bams/1183507843
46. Shan, Z., Azim, T., Neamtiu, I.: Finding resume and restart errors in android applications. In: Visser, E., Smaragdakis, Y. (eds.) Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, 30 October–4 November 2016, pp. 864–880. ACM (2016). https://doi.org/10.1145/2983990.2984011, http://doi.acm.org/10.1145/2983990.2984011
47. Solar-Lezama, A.: Program sketching. STTT **15**(5–6), 475–495 (2013). https://doi.org/10.1007/s10009-012-0249-7
48. Trinkle, R.: Reflex-frp (2017). https://github.com/reflex-frp/reflex
49. Vechev, M.T., Yahav, E., Yorsh, G.: Abstraction-guided synthesis of synchronization. STTT **15**(5–6), 413–431 (2013). https://doi.org/10.1007/s10009-012-0232-3
50. Wang, X., Dillig, I., Singh, R.: Synthesis of data completion scripts using finite tree automata. PACMPL 1(OOPSLA), 62:1–62:26 (2017). https://doi.org/10.1145/3133886, http://doi.acm.org/10.1145/3133886
51. Winograd-Cort, D.: Effects, Asynchrony, and Choice in Arrowized Functional Reactive Programming. Ph.D. thesis, Yale University, December 2015. http://www.danwc.com/s/dwc-yale-formatted-dissertation.pdf
52. Wongpiromsarn, T., Topcu, U., Murray, R.M.: Synthesis of control protocols for autonomous systems. Unmanned Syst. **1**(01), 21–39 (2013)

53. Yallop, J., Liu, H.: Causal commutative arrows revisited. In: Mainland [36], pp. 21–32. https://doi.org/10.1145/2976002.2976019, http://doi.acm.org/10.1145/2976002.2976019

Run-Time Optimization for Learned Controllers Through Quantitative Games

Guy Avni[1(✉)], Roderick Bloem[2], Krishnendu Chatterjee[1], Thomas A. Henzinger[1], Bettina Könighofer[2], and Stefan Pranger[2]

[1] IST Austria, Klosterneuburg, Austria
guy.avni@ist.ac.at
[2] TU Graz, Graz, Austria

Abstract. A controller is a device that interacts with a plant. At each time point, it reads the plant's state and issues commands with the goal that the plant operates optimally. Constructing optimal controllers is a fundamental and challenging problem. Machine learning techniques have recently been successfully applied to train controllers, yet they have limitations. Learned controllers are monolithic and hard to reason about. In particular, it is difficult to add features without retraining, to guarantee any level of performance, and to achieve acceptable performance when encountering untrained scenarios. These limitations can be addressed by deploying quantitative run-time *shields* that serve as a proxy for the controller. At each time point, the shield reads the command issued by the controller and may choose to alter it before passing it on to the plant. We show how optimal shields that interfere as little as possible while guaranteeing a desired level of controller performance, can be generated systematically and automatically using reactive synthesis. First, we abstract the plant by building a stochastic model. Second, we consider the learned controller to be a black box. Third, we measure *controller performance* and *shield interference* by two quantitative run-time measures that are formally defined using weighted automata. Then, the problem of constructing a shield that guarantees maximal performance with minimal interference is the problem of finding an optimal strategy in a stochastic 2-player game "controller versus shield" played on the abstract state space of the plant with a quantitative objective obtained from combining the performance and interference measures. We illustrate the effectiveness of our approach by automatically constructing lightweight shields for learned traffic-light controllers in various road networks. The shields we generate avoid liveness bugs, improve controller performance in untrained and changing traffic situations, and add features to learned controllers, such as giving priority to emergency vehicles.

1 Introduction

The *controller synthesis* problem is a fundamental problem that is widely studied by different communities [42,44]. A controller is a device that interacts with a *plant*. In each point in time it reads the plant's state, e.g., given by sensor reading, and issues

This research was supported in part by the Austrian Science Fund (FWF) under grants S114 (RiSE/SHiNE), Z211-N23 (Wittgenstein Award), and M 2369-N33 (Meitner fellowship).

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 630–649, 2019.
https://doi.org/10.1007/978-3-030-25540-4_36

a command based on the state. The controller should guarantee that the plant operates correctly or optimally with respect to some given specification. As a running example, we consider a traffic light controller for a road intersection (see Fig. 1). The state of the plant refers to the state of the roads leading to the junction; namely, the positions of the cars, their speeds, their sizes, etc. A controller command consists of a light configuration for the junction in the next time frame. Specifications can either be qualitative, e.g., "it should never be the case that a road with an empty queue gets a green light", or quantitative, e.g., "the cost of a controller is the average waiting times of the cars in the junction".

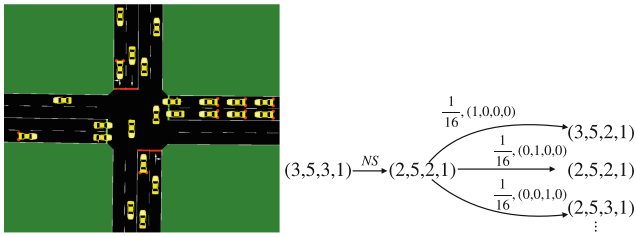


Fig. 1. On the left, a concrete state depicted in the traffic simulator SUMO. On the right, we depict the corresponding abstract state with queues cut off at $k = 5$, and some outgoing transitions. Upon issuing action North-South, a car is evicted from each of the North-South queues. Then, we choose uniformly at random, out of the 16 possible options, the incoming cars to the queues, update the state, and cutoff the queues at k (e.g., when a car enters from East, the queue stays 5).

A challenge in controller synthesis is that, since the number of possible plant readings is huge, it is computationally demanding to find an optimal command, given a plant state. Machine learning is a prominent approach to make decisions based on large amounts of collected data [28,37]. It is widely successful in practice and takes an integral part in the design process of various systems. Machine learning has been successfully applied to train controllers [15,33,34] and specifically controllers for traffic control [20,35,39].

A shortcoming of machine-learning techniques is that the controllers that are produced are black-box devices that are hard to reason about and modify without a complete re-training. It is thus challenging, for example, to obtain worst-case guarantees about the controller, which is particularly important in safety-critical settings. Attempts to address this problem come from both the formal methods community [46], where verification of learned systems is extensively studied [24,29], and the machine-learning community, where guarantees are added during the training process using reward engineering [13,18] or by modifying the exploration process [11,19,38]. Both approaches require expertise in the respective field and suffer from limitations such as scalability for the first, and intricacy and robustness issues, for the second. Moreover, both techniques were mostly studied for safety properties.

Another shortcoming of machine-learning techniques is that they require expertise and a fine-tuning of parameters. It is difficult, for example, to train controllers that are

robust to plant behaviors, e.g., a controller that has been trained on uniform traffic congestion meeting rush-hour traffic, which can be significantly different and can cause poor performance. Also, it is challenging to add features to a controller without retraining, which is both costly and time consuming. These can include permanent features, e.g., priority to public transport, or temporary changes, e.g., changes due to an accident or construction. Again, since the training process is intricate, adding features during training can have unexpected effects.

In this work, we use quantitative *shields* to deal with the limitations of learned or any other black-box controllers. A shield serves as a proxy between the controller and the plant. In each point in time, as before, the controller reads the state of the plant and issues a command. Rather than directly feeding the command to the plant, the shield first reads it along with an abstract plant state. The shield can then choose to keep the controller's command or alter it, before issuing the command to the plant. The concept of shields was first introduced in [30], where shields for safety properties were considered and with a qualitative notion of interference: a shield is only allowed to interfere when a controller error occurs, which is only well-defined when considering safety properties. We elaborate on other shield-like approaches in the Sect. 1.1.

Our goal is to automatically synthesize shields that optimize quantitative measures for black-box controllers. We are interested in synthesizing lightweight shields. We assume that the controller performs well on average, but has no worst-case guarantees. When combining the shield and the controller, intuitively, the controller should be active for the majority of the time and the shield intervenes only when it is required. We formalize the plant behavior as well as the interference cost using quantitative measures. Unlike safety objectives, where it is clear when a shield must interfere, with quantitative objectives, a non-interference typically does not have a devastating effect. It is thus challenging to decide, at each time point, whether the shield should interfere or not; the shield needs to balance the cost of interfering with the decrease in performance of not interfering. Automatic synthesis of shields is thus natural in this setting.

We elaborate on the two quantitative measures we define. The interaction between the plant, controller, and shield gives rise to an infinite sequence over $C \times \Gamma \times \Gamma$, where C is a set of plant states and Γ is a set of allowed actions. A triple $\langle c, \gamma_1, \gamma_2 \rangle$ means that the plant is in state c, the controller issues command γ_1, and the shield (possibly) alters it to γ_2. We use *weighted automata* to assign costs to infinite traces, which have proven to be a convenient, flexible, and robust quantitative specification language [14]. Our *behavioral score* measures the performance of the plant and it is formally given by a weighted automaton that assigns scores to traces over $C \times \Gamma$. Boolean properties are a special case, which include *safety* properties, e.g., "an emergency vehicle should always get a green light", and *liveness*, e.g., "a car waiting in a queue eventually gets the green light". An example of a quantitative score is the long-run average of the waiting times of the vehicles in the city. A second score measures the *interference* of a shield with a controller. It is given by a weighted automaton over the alphabet $\Gamma \times \Gamma$. A simple example of an interference score charges the shield 1 for every change of action and charges 0 when no change is made. Then, the score of an infinite trace can be phrased as the ratio of the time that the shield interferes. Using weighted automata we can specify more involved scores such as different charges for different types of alterations or even

charges that depend on the past, e.g., altering the controller's command twice in a row is not allowed.

Given a probabilistic plant model and a formal specification of behavioral and interference scores, the problem of synthesizing an optimal shield is well-defined and can be solved by game theory. While the game-based techniques we use are those of discrete-event controller synthesis [3] in a stochastic setting with quantitative objectives, our set-up is quite different. In traditional controller synthesis, there are two entities; the controller and the adversarial plant. The goal is to synthesize a controller offline. In our setting, there are three entities: the plant, whose behavior we model probabilistically, the controller, which we treat as a black-box and model as an adversary, and the shield, which we synthesize. Note that the shield's synthesis procedure is done offline but it makes online decisions when it operates together with the controller and plant. Our plant model is formally given by a *Markov decision process* which is a standard model with which one models lack of knowledge about the plant using probability (see Fig. 1 and details in Example 1). The game is played on the MDP by two players; a shield and a controller, where the quantitative objective is given by the two scores. An optimal shield is then extracted from an optimal strategy for the shield player. The game we construct admits memoryless optimal strategies, thus the size of the shield is proportional to the size of the abstraction of the plant. In addition, it is implemented as a look-up table for actions in every state. Thus, the runtime overhead is a table look-up and hence negligible.

We experiment with our framework by constructing shields for traffic lights in a network of roads. Our experimental results illustrate the usefulness of the framework. We construct shields that consistently improve the performance of controllers, especially when exhibiting behavior that they are not trained on, but, more surprising, also while exhibiting trained behavior. We show that the use of a shield reduces variability in performance among various controllers, thus when using a shield, the choice of the parameters used in the training phase becomes less acute. We show how a shield can be used to add the functionality of prioritizing public transport as well as local fairness to a controller, both without re-training the controller. In addition, we illustrate how shields can add worst-case guarantees on liveness without a costly verification of the controller.

1.1 Related Work

A shield-like approach to adding safety to systems is called *runtime assurance* [47], and has applications, for example, in control of robotics [41] and drones [12]. In this framework, a switching mechanism alternates between running a high-performance system and a provably safe one. These works differ from ours since they consider safety specifications. As mentioned earlier, a challenge with quantitative specifications is that, unlike safety specifications, a non-interference typically does not have a devastating effect, thus it is not trivial to decide when and to what extent to interfere.

Another line of work is *runtime enforcement*, where an enforcer monitors a program that outputs events and can either terminate the program once it detects an error [45], or alter the event in order to guarantee, for example, safety [21], richer qualitative objectives [16], or privacy [26,49]. The similarities between an enforcer and a shield is in

their ability to alter events. The settings are quite different, however, since the enforced program is not reactive whereas we consider a plant that receives commands.

Recently, formal approaches were proposed in order to restrict the exploration of the learning agent such that a set of logically constraints are always satisfied. This method can support other properties beyond safety, e.g., probabilistic computation tree logic (PCTL) [25,36], linear temporal logic (LTL) [1], or differential dynamic logic [17]. To the best of our knowledge, quantitative specifications have not yet been considered. Unlike these approaches, we consider the learned controller as a black box, thus our approach is particularly suitable for machine learning non-experts.

While MDPs and partially-observable MDPs have been widely studied in the literature w.r.t. to quantitative objectives [27,43], our framework requires the interaction of two players (the shield and the black-box controller) and we use game-theoretic framework with quantitative objectives for our solution.

2 Definitions and Problem Statement

2.1 Plants, Controllers, and Shields

The interaction with a *plant* over a concrete set of states C is carried out using two functionalities: PLANT.GETSTATE returns the plant's current state and PLANT.ISSUECOMMAND issues an action from a set Γ. Once an action is issued, the plant updates its state according to some unknown transition function. At each point in time, the *controller* reads the state of the plant and issues a command. Thus, it is a function from a history in $(C \times \Gamma)^* \cdot C$ to Γ.

Informally, a *shield* serves as a proxy between the controller and the plant. In each time point, it reads the controller's issued action and can choose an alternative action to issue to the plant. We are interested in light-weight shields that add little or no overhead to the controller, thus the shield must be defined w.r.t. an abstraction of the plant, which we define formally below.

Abstraction. An abstraction is a *Markov decision process* (MDP, for short) is $\mathcal{A} = \langle \Gamma, A, a_0, \delta \rangle$, where Γ is a set of actions, A is a set of abstract plant states, $a_0 \in A$ is an initial state, and $\delta : A \times \Gamma \to [0,1]^A$ is a probabilistic transition function, i.e., for every $a \in A$ and $\gamma \in \Gamma$, we have $\sum_{a' \in A} \delta(a,\gamma)(a') = 1$. The probabilities in the abstraction model our lack of knowledge of the plant, and we assume that they reflect the behavior exhibited by the plant. A *policy* f is a function from a finite history of states in A^* to the next action in Γ, thus it gives rise to a probabilistic distribution $\mathcal{D}(f)$ over infinite sequences over A.

Example 1. Consider a plant that represents a junction with four incoming directions (see Fig. 1). We describe an abstraction $\mathcal{A}$ for the junction that specifies how many cars are waiting in each queue, where we cut off the count at a parameter $k \in \mathbb{N}$. Formally, an abstract state is a vector in $\{0, \ldots, k\}^4$, where the indices respectively represent the North, East, South, and West queues. The larger k is, the closer the abstraction is to the concrete plant. The set of possible actions represent the possible light directions in the junction $\{\text{NS}, \text{EW}\}$. The abstract transitions estimate the plant behavior, and

we describe them in two steps. Consider an abstract state $a = (a_1, a_2, a_3, a_4)$ and suppose the issued action is NS, where the case of EW is similar. We allow a car to cross the junction from each of the North and South queues and decrease the two queues. Let $a' = (\max\{0, a_1 - 1\}, a_2, \max\{0, a_3 - 1\}, a_4)$. Next, we probabilistically model incoming cars to the queues as follows. Consider a vector $\langle i_1, i_2, i_3, i_4\rangle \in \{0, 1\}^4$ that represents incoming cars to the queues. Let a'' be such that, for $1 \leq j \leq 4$, we add i_j to the j-th queue and trim at k, thus $a''_j = \min\{a'_j + i_j, k\}$. Then, in $\mathcal{A}$, when performing action NS in a, we move to a'' with the uniform probability $1/16$. □

We define shields formally. Let Γ be a set of commands, M a set of memory states, C and A be a set of concrete and abstract states, respectively, and let $\alpha : C \to A$ be a mapping between the two. A shield is a function $\textsc{Shield} : A \times M \times \Gamma \to \Gamma \times M$ together with an initial memory state $m_0 \in M$. We use $\textsc{Plant}$ to refer to the plant, which, recall, has two functionalities: reading the current state and issuing a command from Γ. Let $\textsc{Cont}$ be a controller, which has a single functionality: given a history of plant states, the controller issues the command to issue to the plant. The interaction of the components is captured in the following pseudo code:

$m \leftarrow m_0 \in M$ and $\pi \leftarrow$ empty sequence.
while true **do**
  $c \leftarrow \textsc{Plant.GetState}() \in C$
  $\gamma \leftarrow \textsc{Cont.GetCommand}(\pi \cdot c)$
  $a = \alpha(c) \in A$ // generate abstract state for shield
  $\gamma', m' \leftarrow \textsc{Shield}(a, \gamma, m)$
  $\textsc{Plant.IssueCommand}(\gamma')$
  $m \leftarrow m'$ // update shield memory state
  $\pi \leftarrow \pi \cdot \langle c, \gamma'\rangle$ // update plant history
end while

2.2 Quantitative Objectives for Shields

We are interested in two types of performance measures for shields. The *behavioral measure* quantifies the quality of the plant's behavior when operated with a controller and shield. The *interference measure* quantifies the degree to which a shield interferes with the controller. Formally, we need to specify values for infinite sequences, and we use *weighted automata*, which are a convenient model to express such values.

Weighted Automata. A weighted automaton is a function from infinite strings to values. Technically, a weighted automaton is similar to a standard automaton only that the transitions are labeled, in addition to letters, with numbers (weights). Unlike standard automata in which a run is either accepting or rejecting, a run in a weighted automaton has a value. We focus on limit-average automata in which the value is the limit average of the running sum of weights that it traverses. Formally, a weighted automaton is $\mathcal{W} = \langle \Sigma, Q, q_0, \Delta, cost\rangle$, where Σ is a finite alphabet, Q is a finite set of states, $\Delta \subseteq (Q \times \Sigma \times Q)$ is a deterministic transition relation, i.e., for every $q \in Q$ and $\sigma \in \Sigma$, there is at most one $q' \in Q$ with $\Delta(q, \sigma, q')$, and $cost : \Delta \to \mathbb{Q}$ specifies costs for transitions. A *run* of $\mathcal{W}$ on an infinite word $\sigma = \sigma_1, \sigma_2, \ldots$ is $r = r_0, r_1, \ldots \in Q^\omega$

such that $r_0 = q_0$ and, for $i \geq 1$, we have $\Delta(r_{i-1}, \sigma_i, r_i)$. Note that $\mathcal{W}$ is deterministic so there is at most one run on every word. The value that $\mathcal{W}$ assigns to σ is $\liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} cost(r_{i-1}, \sigma_i, r_i)$.

Behavioral Score. A *behavioral* score measures the quality of the behavior that the plant exhibits. It is given by a weighed automaton over the alphabet $A \times \Gamma$, thus it assigns real values to infinite sequences over $A \times \Gamma$. In our experiments, we use a *concrete* behavioral score, which assigns values to infinite sequences over $C \times \Gamma$. We compare the performance of the plant with various controllers and shields w.r.t. the concrete score rather than the abstract score. With a weighted automaton we can express costs that change over time: for example, we can penalize traffic lights that change frequently.

Interference Score. The second score we consider measures the interference of the shield with the controller. An *interference* score is given by a weighted automaton over the alphabet $\Gamma \times \Gamma$. With a weighted automaton we can express costs that change over time: for example, interfering once costs 1 and any successive interference costs 2, thus we reward the shield for short interferences.

From Shields and Controllers to Policies. Consider an abstraction MDP $\mathcal{A}$. To ensure worst-case guarantees, we treat the controller as an adversary for the shield. Let SHIELD be a shield with memory set M and initial memory state m_0. Intuitively, we find a policy in $\mathcal{A}$ that represents the interaction of SHIELD with a controller that maximizes the cost incurred. Formally, an *abstract controller* is a function $\chi : A^* \to \Gamma$. The interaction between SHIELD and χ gives rise to a policy $pol(\text{SHIELD}, \chi)$ in $\mathcal{A}$, which, recall, is a function from A^* to Γ. We define $pol(\text{SHIELD}, \chi)$ inductively as follows. Consider a history $\pi \in A^*$ that ends in $a \in A$, and suppose the current memory state of SHIELD is $m \in M$. Let $\gamma = \chi(\pi)$ and let $\langle \gamma', m' \rangle = \text{SHIELD}(\gamma, a, m)$. Then, the action that the policy $pol(\text{SHIELD}, \chi)$ assigns is γ', and we update the memory state to be m'.

Problem Definition; Quantitative Shield Synthesis Consider an abstraction MDP $\mathcal{A}$, a behavioral score BEH, an interference score INT, both given as weighted automata, and a factor $\lambda \in [0, 1]$ with which we weigh the two scores. Our goal is to find an *optimal shield* w.r.t. these inputs as we define below. Consider a shield SHIELD with memory set M. Let X be the set of abstract controllers. For SHIELD and $\chi \in X$, let $\mathcal{D}(\text{SHIELD}, \chi)$ be the probability distribution over $A \times \Gamma \times \Gamma$ that the policy $pol(\text{SHIELD}, \chi)$ gives rise to. The *value* of SHIELD, denoted $val(\text{SHIELD})$, is $\sup_{\chi \in X} \mathbb{E}_{r \sim \mathcal{D}(\text{SHIELD}, \chi)}[\lambda \cdot \text{INT}(r) + (1 - \lambda) \cdot \text{BEH}(r)]$. An *optimal shield* is a shield whose value is $\inf_{\text{SHIELD}} val(\text{SHIELD})$.

Remark 1 **(Robustness and flexibility).** The problem definition we consider allows quantitative optimization of shields w.r.t. two dimensions of quantitative measures. Earlier works have considered shields but mainly with respect to Boolean measures in both dimensions. For example, in [30], shields for safety behavioral measures were constructed with a Boolean notion of interference, as well as a Boolean notion of shield correctness. In contrast we allow quantitative objectives in both dimensions which presents a much more general and robust framework. For example, the first measure of correctness can be quantitative and minimize the error rate, and the second measure can allow

shields to correct but minimize the long-run average interference. Both of the above allows the shield to be flexible. Moreover, tuning the parameter λ allows flexible trade-off between the two.

We allow a robust class of quantitative specifications using weighted automata, which have been already established as a robust specification framework. Any automata model can be used in the framework, not necessarily the ones we use here. For example, weighted automata that discount the future or process only finite-words are suitable for planning purposes [32]. Thus our framework is a very robust and flexible framework for quantitative shield synthesis. □

2.3 Examples

In Remark 1 we already discussed the flexibility of the framework. We now present concrete examples of instantiations of the optimization problem above on our running example, which illustrate how quantitative shields can be used to cope with limitations of learned controllers.

Dealing with Unexpected Plant Behavior; Rush-Hour Traffic. Consider the abstraction described in Example 1, where each abstract state is a 4-dimensional vector that represents the number of waiting cars in each direction. The behavioral score we use is called the *max queue*. It charges an abstract state $a \in \{0, \ldots, k\}^4$ with the size of the maximal queue, no matter what the issued action is, thus $cost_{\text{BEH}}(a) = \max_{i \in \{1,2,3,4\}} a_i$. A shield that minimizes the max-queue cost will prioritize the direction with the largest queue. For the interference score, we use a score that we call the *basic* interference score; we charge the shield 1 whenever it changes the controller's action and otherwise we charge it 0, and take the long-run average of the costs. Recall that in the construction in Example 1, we chose uniformly at random the vector of incoming cars. Here, in order to model rush-hour traffic, we use a different distribution, where we let p_j be the probability that a car enters the j-th queue. Then, the probability of a vector $\langle i_1, i_2, i_3, i_4 \rangle \in \{0, 1\}^4$ is $\prod_{1 \leq j \leq 4} (p_j \cdot i_j + (1 - p_j) \cdot (1 - i_j))$. To model a higher load traveling on the North-South route, we increase p_1 and p_3 beyond 0.5.

Weighing Different Goals; Local Fairness. Suppose the controller is trained to maximize the number of cars passing a *city*. Thus, it aims to maximize the speed of the cars in the city and prioritizes highways over farm roads. A secondary objective for a controller is to minimize local queues. Rather than adding this objective in the training phase, which can have an un-expected outcome, we can add a local shield for each junction. To synthesize the shield, we use the same abstraction and basic interference score as in the above. The behavioral score we use charges an abstract state $a \in \{0, \ldots, k\}^4$ with difference $|(a_1 + a_3) - (a_2 + a_4)|$, thus the greater the inequality between the two waiting directions, the higher the cost.

Adding Features to the Controller; Prioritizing Public Transport. Suppose a controller is trained to increase throughput in a junction. After the controller is trained, a designer wants to add a functionality to the controller that prioritizes buses over personal vehicles. That is, if a bus is waiting in the North direction, and no bus is waiting in either the East or West directions, then the light should be North-South, and the other

cases are similar. The abstraction we use is simpler than the ones above since we only differentiate between a case in which a bus is present or not, thus the abstract states are $\{0,1\}^4$, where the indices represent the directions clockwise starting from North. Let $\gamma = \text{NS}$. The behavioral cost of a state a is 1 when $a_2 = a_4 = 0$ and $a_1 = 1$ or $a_3 = 1$. The interference score we use is the basic one. A shield guarantees that in the long run, the specification is essentially never violated.

3 A Game-Theoretic Approach to Quantitative Shield Synthesis

In order to synthesize optimal shields we construct a two-player stochastic game [10], where we associate Player 2 with the shield and Player 1 with the controller. The game is defined on top of an abstraction and the players' objectives are given by the two performance measures. We first formally define stochastic games, then we construct the shield synthesis game, and finally show how to extract a shield from a strategy for Player 2.

Stochastic Graph Games. The game is played on a graph by placing a token on a vertex and letting the players move it throughout the graph. For ease of presentation, we fix the order in which the players move: first, Player 1, then Player 2, and then "Nature", i.e., the next vertex is chosen randomly. Edges have costs, which, again for convenience, appear only on edges following Player 2 moves. Formally, a two-player stochastic graph-game is $\langle V_1, V_2, V_N, E, \Pr, \mathit{cost}\rangle$, where $V = V_1 \cup V_2 \cup V_N$ is a finite set of vertices that is partitioned into three sets, for $i \in \{1,2\}$, Player i controls the vertices in V_i and "Nature" controls the vertices in V_N, $E \subseteq (V_1 \times V_2) \cup (V_2 \times V_N)$ is a set of deterministic edges, $\Pr : V_N \times V_1 \to [0,1]$ is a probabilistic transition function, and $\mathit{cost} : (V_2 \times V_N) \to \mathbb{Q}$. Suppose the token reaches $v \in V$. If $v \in V_i$, for $i \in \{1,2\}$, then Player i chooses the next position of the token $u \in V$, such that $E(v,u)$. If $v \in V_N$, then the next position is chosen randomly; namely, the token moves to $u \in V$ with probability $\Pr[v,u]$.

The game is a *zero-sum game*; Player 1 tries to maximize the expected long-run average of the accumulated costs, and Player 2 tries to minimize it. A *strategy* for Player i, for $i \in \{1,2\}$, is a function that takes a history in $V^* \cdot V_i$ and returns the next vertex to move the token to. The games we consider admit *memoryless* optimal strategies, thus it suffices to define a Player i strategy as a function from V_i to V. We associate a *payoff* with two strategies f_1 and f_2, which we define next. Given f_1 and f_2, it is not hard to construct a Markov chain $\mathcal{M}$ with states V_N and with weights on the edges: for $v, u \in V_N$, the probability of moving from v to u in $\mathcal{M}$ is $\Pr_{\mathcal{M}}[v,u] = \sum_{w \in V_1 : f_2(f_1(w))=u} \Pr[v,w]$ and the cost of the edge is $\mathit{cost}_{\mathcal{M}}(v,u) = \sum_{w \in V_1 : f_2(f_1(w))=u} \Pr[v,w] \cdot \mathit{cost}(f_1(w),u)$. The *stationary distribution* s_v of a vertex $v \in V_N$ in $\mathcal{M}$ is a well known concept [43] and it intuitively measures the long-run average time that is spend in v. The payoff w.r.t. f_1 and f_2, denoted $\text{payoff}(f_1,f_2)$ is $\sum_{v,u \in V_N} s_v \cdot \Pr_{\mathcal{M}}[v,u] \cdot \mathit{cost}_{\mathcal{M}}(v,u)$. The payoff of a strategy is the payoff it guarantees against any strategy of the other player, thus $\text{payoff}(f_1) = \inf_{f_2} \text{payoff}(f_1,f_2)$. A strategy is *optimal* for Player 1 if it achieves the optimal payoff, thus f is optimal if $\text{payoff}(f) = \sup_{f_1} \text{payoff}(f_1)$. The definitions for Player 2 are dual.

Constructing the Synthesis Game. Consider an abstraction MDP $\mathcal{A} = \langle \Gamma, A, a_0, \delta \rangle$, weighted automata for the behavioral score $\textsc{Beh} = \langle A \times \Gamma, Q_{\textsc{Beh}}, q_0^{\textsc{Beh}}, \Delta_{\textsc{Beh}}, cost_{\textsc{Beh}} \rangle$ and interference score $\textsc{Int} = \langle \Gamma \times \Gamma, Q_{\textsc{Int}}, q_0^{\textsc{Int}}, \Delta_{\textsc{Int}}, cost_{\textsc{Int}} \rangle$, and a factor $\lambda \in [0, 1]$. We associate Player 1 with the controller and Player 2 with the shield. In each step, the controller first chooses an action, then the shield chooses whether to alter it, and the next state is selected at random. Let $S = A \times Q_{\textsc{Int}} \times Q_{\textsc{Beh}}$. We define $\mathcal{G}_{\mathcal{A},\textsc{Beh},\textsc{Int},\lambda} = \langle V_1, V_2, V_N, E, \Pr, cost \rangle$, where

- $V_1 = S$,
- $V_2 = S \times \Gamma$,
- $V_N = S \times \Gamma \times \{N\}$, where the purpose of N is to differentiate between the vertices,
- $E(s, \langle s, \gamma \rangle)$ for $s \in S$ and $\gamma \in \Gamma$, and $E(\langle s, \gamma \rangle, \langle s', \gamma', N \rangle)$ for $s = \langle a, q_1, q_2 \rangle \in S, \gamma, \gamma' \in \Gamma$, and $s' = \langle a, q_1', q_2' \rangle \in S$ s.t. $\Delta_{\textsc{Int}}(q_1, \langle \gamma, \gamma' \rangle, q_1')$ and $\Delta_{\textsc{Beh}}(q_2, \langle a, \gamma' \rangle, q_2')$,
- $\Pr[\langle \langle a, q_1, q_2 \rangle, \gamma, N \rangle, \langle a', q_1, q_2 \rangle] = \delta(a, \gamma)(a')$, and
- for $s = \langle a, q_1, q_2 \rangle$ and $s' = \langle a, q_1', q_2' \rangle$ as in the above, we have $cost(\langle s, \gamma \rangle, \langle s', \gamma', N \rangle) = \lambda \cdot cost_{\textsc{Int}}(q_1, \langle \gamma, \gamma' \rangle, q_1') + (1 - \lambda) \cdot cost_{\textsc{Beh}}(q_2, \langle \gamma', a \rangle, q_2')$.

From Strategies to Shields. Recall that the game $\mathcal{G}_{\mathcal{A},\textsc{Beh},\textsc{Int},\lambda}$ admits memoryless optimal strategies. Consider an optimal memoryless strategy f for Player 2. Thus, given a Player 2 vertex in V_2, the function f returns a vertex in V_N to move to. The shield $\textsc{Shield}_f$ that is associated with f has the memory set $M = Q_{\textsc{Int}} \times Q_{\textsc{Beh}}$ and the initial memory state is $\langle q_0^{\textsc{Int}}, q_0^{\textsc{Beh}} \rangle$. Given an abstract state $a \in A$, a memory state $\langle q_{\textsc{Int}}, q_{\textsc{Beh}} \rangle \in M$, and a controller action $\gamma \in \Gamma$, let $\langle a, q'_{\textsc{Int}}, q'_{\textsc{Beh}}, \gamma' \rangle = f(a, q_{\textsc{Int}}, q_{\textsc{Beh}}, \gamma)$. The shield $\textsc{Shield}_f$ returns the action γ' and the updated memory state $\langle q'_{\textsc{Int}}, q'_{\textsc{Beh}} \rangle$.

Theorem 1. *Given an abstraction $\mathcal{A}$, weighted automata* Beh *and* Int*, and a factor λ, the game $\mathcal{G}_{\mathcal{A},\textsc{Beh},\textsc{Int},\lambda}$ admits optimal memoryless strategies. Let f be an optimal memoryless strategy for Player* 2*. The shield* $\textsc{Shield}_f$ *is an optimal shield w.r.t. $\mathcal{A}$,* Beh*,* Int*, and λ.*

Remark 2 **(Shield size).** Recall that a shield is a function $\textsc{Shield} : A \times \Gamma \times M \to \Gamma \times M$, which we store as a table. The *size* of the shield is the size of the domain, namely the number of entries in the table. Given an abstraction with n_1 states, a set of possible commands Γ, and weighted automata with n_2 and n_3 states, the size of the shield we construct is $n_1 \cdot n_2 \cdot n_3 \cdot |\Gamma|$. □

Remark 3. Our construction of the game can be seen as a two-step procedure: we construct a stochastic game with two mean-payoff objectives, a.k.a. a *two-dimensional* game, where the shield player's goal is to minimize both the behavioral and interference scores separately. We then reduce the game to a "one-dimension" game by weighing the scores with the parameter λ. We perform this reduction for several reasons. First, while multi-dimensional quantitative objectives have been studied in several cases, such as MDPs [4,6,7] and special problems of stochastic games (e.g., almost-sure winning) [2,5,8], there is no general algorithmic solution known for stochastic games with two-dimensional objectives. Second, even for non-stochastic games with

two-dimensional quantitative objectives, infinite-memory is required in general [48]. Finally, in our setting, the parameter λ provides a meaningful tradeoff: it can be associated with how well we value the quality of the controller. If the controller is of poor quality, then we charge the shield less for interference and set λ to be low. On the other hand, for a high-quality controller, we charge the shield more for interferences and set a high value for λ. □

4 Case Study

We experiment with our framework in designing quantitative shields for traffic-light controllers that are trained using reinforcement-learning (RL). We illustrate the usefulness of shields in dealing with limitations of RL as well as providing an intuitive framework to complement RL techniques.

Traffic Simulation. All experiments were conducted using traffic simulator "Simulation of Urban MObility" (SUMO, for short) [31] v0.22 using the SUMO Python API. Incoming traffic in the cities is chosen randomly. The simulations were executed on a desktop computer with a 4 x 2.70 GHz Intel Core i7-7500U CPU, 7.7 GB of RAM running Ubuntu 16.04.

The Traffic Light Controller. We use RL to train a city-wide traffic-signal controller. Intuitively, the controller is aware of the waiting cars in each junction and its actions constitute a light assignment to all the junctions. We train a controller using a deep convolutional Q-network [37]. In most of the networks we test with, there are two controlled junctions. The input vector to the neural network is a 16-dimensional vector, where 8 dimensions represent a junction. For each junction, the first four components state the number of cars approaching the junction and the last four components state the accumulated waiting time of the cars in each one of the lanes. For example, in Fig. 1, the first four components are $(3, 6, 3, 1)$, thus the controller's state is not trimmed at 5. The controller is trained to minimize both the number of cars waiting in the queues and the total waiting time. For each junction i, the controller can choose to set the light to be either NS_i or EW_i, thus the set of possible actions is $\Gamma = \{\text{NS}_1\text{NS}_2, \text{EW}_1\text{NS}_2, \text{NS}_1\text{EW}_2, \text{EW}_1\text{EW}_2\}$.

We use a network consisting of 4 layers: The input layer is a convolutional layer with 16 nodes, the first hidden and the second hidden layers consisting out of 604 nodes and 1166 nodes, respectively. The output layer consists of 4 neurons with linear activation functions, each representing one of the above mentioned actions listed in Γ. The Q-learning uses the learning rate $\alpha = 0.001$ and the discount factor 0.95 for the Q-update and an ϵ-greedy exploration policy. The artificial neural network is built on an open source implementation[1] using Keras [9] and additional optimized functionality was provided by the NumPy [40] library. We train for 100 training epochs, where each epoch is 1500 seconds of simulated traffic, plus 2000 additional seconds in which no new cars are introduced. The total training time of the agent is roughly 1.5 hours. While the RL procedure that we use is simple procedure, it is inspired by standard approaches

[1] https://github.com/Wert1996/Traffic-Optimisation.

to learning traffic controllers and produces controllers that perform relatively well also with no shield.

The Shield. We synthesize a "local" shield for a junction and copy the shield for each junction in the city. Recall that the first step in constructing the synthesis game is to construct an abstraction of the plant, which intuitively represents the information according to which the shield makes its decisions. The abstraction we use is described in Example 1; each state is a 4-dimensional integer in $\{0, \ldots, k\}$, which represents an abstraction of the number of waiting cars in each direction, cut-off by $k \in \mathbb{N}$. As elaborated in the example, when a shield assigns a green light to a direction, we evict a car from the two respectable queues, and select the incoming cars uniformly at random. Regarding objectives, in most of our experiments, the behavioral score we use charges an abstract state $a \in \{0, \ldots, k\}^4$ with $|(a_1 + a_3) - (a_2 + a_4)|$, thus the shield aims to balance the total number of waiting cars per direction. The interference score we use charges the shield 1 for altering the controller's action.

Since we use simple automata for objectives, the size of the shields we use is $|A \times \Gamma|$, where $|\Gamma| = 2$. In our experiments, we cut-off the queues at $k = 6$, which results in a shield of size 2592. The synthesis procedure's running time is in the order of minutes. We have already pointed out that we are interested in small light-weight shields, and this is indeed what we construct. In terms of absolute size, the shield takes $\sim$60 KB versus the controller who takes $\sim$3 MB; a difference of 2 orders of magnitude.

Our synthesis procedure includes a solution to a stochastic mean-payoff game. The complexity of solving such games is an interesting combinatorial problem in NP and coNP (thus unlikely to be NP-hard) for which the existence of a polynomial-time algorithm is major long-standing open problem. The current best-known algorithms are exponential, and even for special cases like turn-based deterministic mean-payoff games or turn-based stochastic games with reachability objectives, no polynomial-time algorithms are known. The algorithm we implemented is called the *strategy iteration* algorithm [22,23] in which one starts with a strategy and iteratively improves it, where each iteration requires polynomial time. While the algorithm's worst-case complexity is exponential, in practice, the algorithm has been widely observed to terminate in a few number of iterations.

Evaluating Performance. Throughout all our experiments, we use a unified and concrete measure of performance: the total waiting time of the cars in the city. Our assumption is that minimizing this measure is the main objective of the designer of the traffic light system for the city. While performance is part of the objective function when training the controller, the other components of the objective are used in order to improve training. Similarly, the behavioral measure we use when synthesizing shields is chosen heuristically in order to construct shields that improve concrete performance.

The Effect of Changing λ. Recall that we use $\lambda \in [0, 1]$ in order to weigh between the behavioral and interference measures of a shield, where the larger λ is, the more the shield is charged for interference. In our first set of experiments, we fix all parameters apart from λ and synthesize shields for a city that has two controllable junctions. In the first experiment, we use a random traffic flow that is similar to the one used in training.

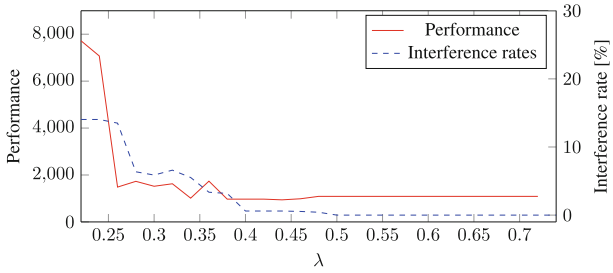


Fig. 2. Results for shields constructed with various λ values, together with a fixed plant and controller, where the simulation traffic distribution matches the one the controller is trained for.

We depict the results of the simulation in Fig. 2. We make several observations on the results below.

Interference. We observe that the ratio of the time that the shield intervenes is low: for most values of λ the ratio is well below 10%. For large values of λ, interference is too costly, and the shields become *trivial*, namely it never alters the actions of the controller. The performance we observe is thus the performance of the controller with no shield. In this set of experiments, we observe that the threshold after which shields become trivial is $\lambda = 0.5$, and for different setups, the threshold changes.

Performance. We observe that performance as function of λ, is a curve-like function. When λ is small, altering commands is cheap, the shield intervenes more frequently, and performance drops. This performance drop is expected: the shield is a simple device and the quality of its routing decisions cannot compete with the trained controller. This drop is also encouraging since it illustrates that our experimental setting is interesting. Surprisingly, we observe that the curve is in fact a paraboloid: for some values, e.g., $\lambda = 0.4$, the shield improves the performance of the controller. We find it unexpected that the shield improves performance even when observing trained behavior, and this performance increase is observed more significantly in the next experiments.

Rush-Hour Traffic. In Fig. 3, we use a shield to add robustness to a controller for behavior it was not trained for. We see a more significant performance gain in this exper-

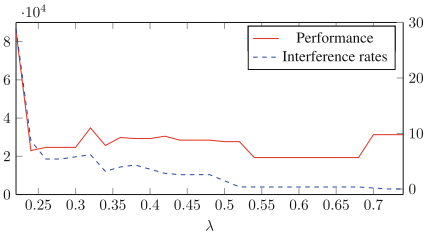


Fig. 3. Similar to Fig. 2 only that the simulation traffic distribution models rush-hour traffic.

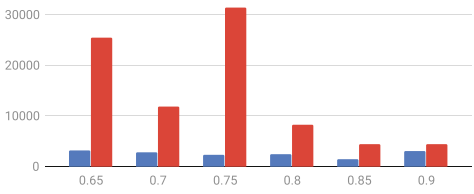


Fig. 4. Comparing the variability in performance of the different controllers, with shield (blue) and without a shield (red). (Color figure online)

iment. We use the controller from the previous experiment, which is trained for uniform car arrival. We simulate it in a network with "rush-hour" traffic, which we model by significantly increasing the traffic load in the North-South direction. We synthesize shields that prefer to evict traffic from the North-South queue over the East-West queue. We achieve this by altering the objective in the stochastic game; we charge the shield a greater penalty for cars waiting in these queues over the other queues. For most values of λ below 0.7, we see a performance gain. Note that the performance of the controller with no shield is depicted on the far right, where the shield is trivial. An alternative approach to synthesize a shield would be to alter the probabilities in the abstraction, but we found that altering the weights results in a better performance gain.

Reducing Variability. Machine learning techniques are intricate, require expertise, and a fine tuning of parameters. This set of experiments show how the use of shields reduces variability of the controllers, and as a result, it reduces the importance of choosing the optimal parameters in the training phase. We fix one of the shields from the first experiment with $\lambda = 0.4$. We observe performance in a city with various controllers, which are trained with varying training parameters, when the controllers are run with and without the shield and on various traffic conditions that sometimes differ from the ones they are trained on.

The city we experiment with consists of a main two-lane road that crosses the city from East to West. The main road has two junctions in which smaller "farm roads" meet the main road. We refer to the *bulk traffic* as the traffic that only "crosses the city"; namely, it flows only on the main road either from East to West or in the opposite direction. For $r \in [0, 1]$, Controller-r is trained where the ratio of the bulk traffic out of the total traffic is r. That is, the higher r is, the less traffic travels on the farm roads. We run simulations in which Controller-r observes bulk traffic $k \in [0, 1]$, which it was not necessarily trained for.

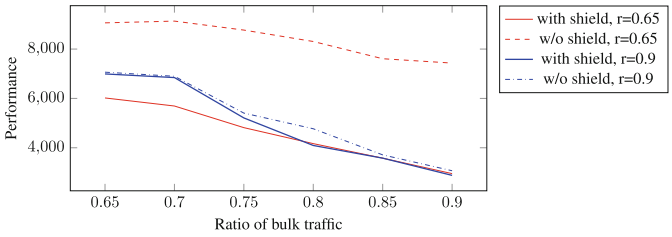


Fig. 5. Results for Controllers-0.65 and 0.9 exhibiting traffic that they are not trained for, with and without a shield. Performance is the total waiting time of the cars in the city.

In Fig. 5, we depict the performance of two controllers for various traffic settings. We observe, in these two controllers as well as the others, that operating with a shield consistently improves performance. The plots illustrate the unexpected behavior of machine-learning techniques: e.g., when run without a shield, Controller-0.9 outperforms Controller-0.65 in all settings, even in the setting 0.65 on which Controller-0.65 was trained on. Thus, a designer who expects a traffic flow of 0.65, would be better

off training with a traffic of 0.9. A shield improves performance and thus reduces the importance of which training data to use.

Measuring Variability. In Fig. 4, we depict the variability in performance between the controllers. The higher the variability is, the more significant it is to choose the right parameters when training the controller. Formally, let $R = \{0.65, 0.7, 0.75, 0.8, 0.85, 0.9\}$. For $r, k \in R$, we let $\text{Perf}(r, k)$ denote the performance (total waiting times) when Controller-r observes bulk traffic k. For each $k \in R$, we depict $\max_{r \in R} \text{Perf}(r, k) - \min_{r' \in R} \text{Perf}(r', k)$, when operating with and without a shield.

Clearly, the variability with a shield is significantly lower than without one. This data shows that when operating with a shield, it does not make much difference if a designer trains a controller with setting r or r'. When operating without a shield, the difference is significant.

Overcoming Liveness Bugs. Finding bugs in learned controllers is a challenging task. Shields bypass the need to find bugs since they treat the controller as a black-box and correct its behavior. We illustrate their usefulness in dealing with liveness bugs. In the same network as in the previous setting, we experiment with a controller whose training process lacked variability. In Fig. 6, we depict the light configuration throughout the experiment on the main road; the horizontal axis represents time, red means a red light for the main road and dually green. Initially, the controller performs well, but roughly half-way through the simulation it hits a bad state after which the light stays red. The shield, with only a few interferences, which are represented with dots, manages to recover the controller from its stuck state. In Fig. 7, we depict the number of waiting cars in the city, which clearly skyrockets once the controller gets stuck. It is evident that initially, the controller performs well. This point highlights that it is difficult to recog-

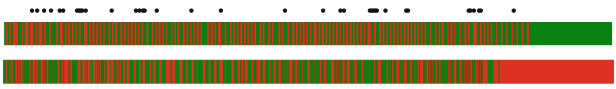


Fig. 6. The light in the East-West direction (the main road) of a junction. On bottom, with no shield the controller is stuck. On top, the shield's interferences are marked with dots.

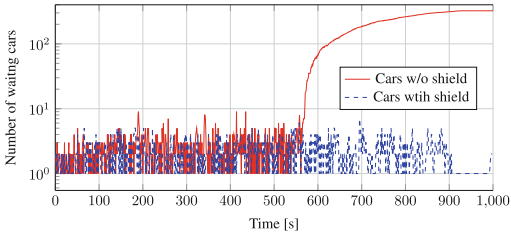


Fig. 7. The total number of waiting cars (log-scale) with and without a shield. Initially, the controller performs well on its own, until it gets stuck and traffic in the city freezes.

nize when a controller has a bug – in order to catch such a bug, a designer would need to find the right simulation and run it half way through before the bug appears.

One way to regain liveness would be to synthesize a shield for the qualitative property "each direction eventually gets a green light". Instead, we use a shield that is synthesized for the quantitative specification as in the previous experiment. The shield, with a total of only 20 alterations is able to recover the controller from the bad state it is stuck in, and traffic flows correctly.

Adding Functionality; Prioritizing Public Transport. Learned controllers are monolithic. Adding functionality to a controller requires a complete re-training, which is time consuming, computationally costly, and requires care; changes in the objective can cause unexpected side effects to the performance. We illustrate how, using a shield, we can add to an existing controller, the functionality of prioritizing public transport.

The abstraction over which the shield is constructed slightly differs from the one used in the other experiments. The abstract state space is the same, namely four-dimensional vectors, though we interpret the entries as the positions of a bus in the respective queue. For example, the state $(0, 3, 0, 1)$ represents no bus in the North queue and a bus which is waiting, third in line, in the East queue. Outgoing edges from an abstract state also differ as they take into account, using probability, that vehicles might enter the queues between buses. For the behavioral score, we charge an abstract state with the sum of its entries, thus the shield is charged whenever buses are waiting and it aims to evict them from the queues as soon as possible.

In Fig. 8, we depict the performance of all vehicles and only buses as a function of the weighing factor λ. The result of this experiment is positive; the predicted behavior is observed. Indeed, when λ is small, interferences are cheap, which increase bus performance at the expense of the general performance. The experiment illustrates that the parameter λ is a convenient method to control the degree of prioritization of buses.

Local Fairness. In this experiment, we add local fairness to a controller that was trained for a global objective. We experiment with a network with four junctions and a city-wide controller, which aims to minimize total waiting times. Figure 9 shows that when the controller is deployed on its own, queues form in the city whereas a shield, which was synthesized as in the first experiments, prevents such local queues from forming.

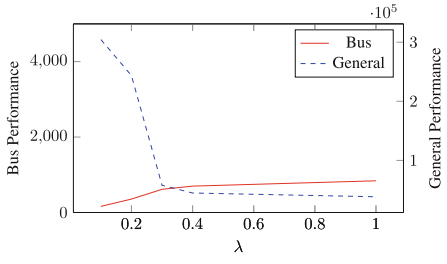


Fig. 8. The waiting time of buses/all vehicles with shields parameterized by λ.

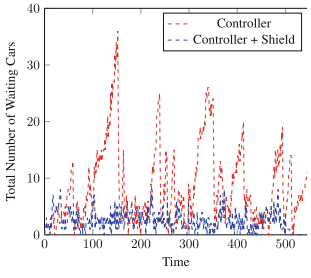


Fig. 9. Comparing the amount of waiting cars with and without a shield.

5 Discussion and Future Work

We suggest a framework for automatically synthesizing quantitative runtime shields to cope with limitations of machine-learning techniques. We show how shields can increase robustness to untrained behavior, deal with liveness bugs without verification, add features without retraining, and decrease variability of performance due to changes in the training parameters, which is especially helpful for machine learning non-experts. We use weighted automata to evaluate controller and shield behavior and construct a game whose solution is an optimal shield w.r.t. a weighted specification and a plant abstraction. The framework is robust and can be applied in any setting where learned or other black-box controllers are used.

We list several directions for further research. In this work, we make no assumptions on the controller and treat it adversarially. Since the controller might have bugs, modelling it as adversarial is reasonable. Though, it is also a crude abstraction since typically, the objectives of the controller and shield are similar. For future work, we plan to study ways to model the spectrum between cooperative and adversarial controllers together with solution concepts for the games that they give rise to.

In this work we make no assumptions on the relationship between the plant and the abstraction. While the constructed shields are optimal w.r.t. the given abstraction, the scores they guarantee w.r.t. the abstraction do not imply performance guarantees on the plant. To be able to produce performance guarantees on the concrete plant, we need guarantees on the relationship between the plant its abstraction. For future work, we plan to study the addition of such guarantees and how they affect the quality measures.

References

1. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: AAAI. AAAI Press (2018)
2. Basset, N., Kwiatkowska, M.Z., Wiltsche, C.: Compositional strategy synthesis for stochastic games with multiple objectives. Inf. Comput. **261**(Part), 536–587 (2018)
3. Bloem, R., Chatterjee, K., Jobstmann, B.: Graph games and reactive synthesis. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 921–962. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_27
4. Brázdil, T., Brozek, V., Chatterjee, K., Forejt, V., Kucera, A.: Two views on multiple mean-payoff objectives in Markov decision processes. Log. Methods Comput. Sci. **10**(1) (2014). https://lmcs.episciences.org/1156
5. Chatterjee, K., Doyen, L.: Perfect-information stochastic games with generalized mean-payoff objectives. In: Proceedings of the 31st LICS, pp. 247–256 (2016)
6. Chatterjee, K., Kretínská, Z., Kretínský, J.: Unifying two views on multiple mean-payoff objectives in Markov decision processes. Log. Methods Comput. Sci. **13**(2) (2017). https://lmcs.episciences.org/3757
7. Chatterjee, K., Majumdar, R., Henzinger, T. A.: Markov decision processes with multiple objectives. In: Proceedings of the 23rd STACS, pp. 325–336 (2006)
8. Chen, T., Forejt, V., Kwiatkowska, M. Z., Simaitis, A., Trivedi, A., Ummels, M.: Playing stochastic games precisely. In: Proceedings of the 23rd CONCUR, pp. 348–363 (2012)
9. Chollet, F.: keras (2015). https://github.com/fchollet/keras

10. Condon, A.: On algorithms for simple stochastic games. In: Proceedings of the DIMACS, pp. 51–72 (1990)
11. Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C., Tassa, Y.: Safe exploration in continuous action spaces. coRR, abs/1801.08757 (2017). arXiv:1801.08757
12. Desai, A., Ghosh, S., Seshia, S. A., Shankar, N., Tiwari, A.: SOTER: programming safe robotics system using runtime assurance. coRR, abs/1808.07921 (2018). arXiv:1808.07921
13. Dewey, D.: Reinforcement learning and the reward engineering principle. In: 2014 AAAI Spring Symposium Series (2014)
14. Droste, M., Kuich, W., Vogler, H.: Handbook of Weighted Automata. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01492-5
15. Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. In: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, 19–24 June 2016, pp. 1329–1338 (2016)
16. Falcone, Y., Mounier, L., Fernandez, J.-C., Richier, J.-L.: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Formal Methods Syst. Des. **38**(3), 223–262 (2011)
17. Fulton, N., Platzer, A.: Safe reinforcement learning via formal methods: toward safe control through proof and learning. In: AAAI. AAAI Press (2018)
18. García, J., Fernández, F.: A comprehensive survey on safe reinforcement learning. J. Mach. Learn. Res. **16**, 1437–1480 (2015)
19. Geibel, P.: Reinforcement learning for MDPs with constraints. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 646–653. Springer, Heidelberg (2006). https://doi.org/10.1007/11871842_63
20. Genders, W., Razavi, S.: Asynchronous n-step q-learning adaptive traffic signal control. J. Intell. Trans. Syst. **23**(4), 319–331 (2019)
21. Hamlen, K.W., Morrisett, J.G., Schneider, F.B.: Computability classes for enforcement mechanisms. ACM Trans. Program. Lang. Syst. **28**(1), 175–205 (2006)
22. Hoffman, A.J., Karp, R.M.: On nonterminating stochastic games. Manag. Sci. **12**(5), 359–370 (1966)
23. Howard, A.R.: Dynamic Programming and Markov Processes. MIT Press, Cambridge (1960)
24. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Proceedings of the 29th CAV, pp. 3–29 (2017)
25. Jansen, N., Könighofer, B., Junges, S., Bloem, R.: Shielded decision-making in MDPs. CoRR, arXiv:1807.06096 (2018)
26. Ji, Y., Lafortune, S.: Enforcing opacity by publicly known edit functions. In: 56th IEEE Annual Conference on Decision and Control, CDC 2017, Melbourne, Australia, 12–15 December 2017, pp. 4866–4871 (2017)
27. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artif. Intell. **101**(1), 99–134 (1998)
28. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. JAIR **4**, 237–285 (1996)
29. Katz, G., Barrett, C.W., Dill, C. W., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Proceedings of the 29th CAV, pp. 97–117 (2017)
30. Könighofer, B., Alshiekh, M., Bloem, R., Humphrey, L., Könighofer, R., Topcu, U., Wang, C.: Shield synthesis. Formal Methods Syst. Des. **51**(2), 332–361 (2017)
31. Krajzewicz, D., Erdmann, J., Behrisch, M., Bieker, L.: Recent development and applications of SUMO - Simulation of Urban MObility. Int. J. Adv. Syst. Meas. **5**(3&4), 128–138 (2012)

32. Lahijanian, M., Almagor, S., Fried, D., Kavraki, L.E., Vardi, M.Y.: This time the robot settles for a cost: a quantitative approach to temporal logic planning with partial satisfaction. In: Proceedings of the 29th AAAI, pp. 3664–3671 (2015)
33. Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., Quillen, D.: Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. I. J. Robot. Res. **37**(4–5), 421–436 (2018)
34. Lillicrap, T.P.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
35. Mannion, P., Duggan, J., Howley, E.: An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In: McCluskey, T.L., Kotsialos, A., Müller, J.P., Klügl, F., Rana, O., Schumann, R. (eds.) Autonomic Road Transport Support Systems. AS, pp. 47–66. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-25808-9_4
36. Mason, G., Calinescu, R., Kudenko, D., Banks, A.: Assured reinforcement learning with formally verified abstract policies. In: Proceedings of the 9th International Conference on Agents and Artificial Intelligence, ICAART 2017, Porto, Portugal, 24–26 February 2017, vol. 2, pp. 105–117 (2017)
37. Mnih, V.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
38. Moldovan, T.M., Abbeel, P.: Safe exploration in Markov decision processes. In: Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 – July 1, 2012 (2012)
39. Mousavi, S.S., Schukat, M., Howley, E.: Traffic light control using deep policy-gradient and value-function-based reinforcement learning. IET Intell. Trans. Syst. **11**(7), 417–423 (2017)
40. Oliphant, T.E.: Guide to NumPy, 2nd edn. CreateSpace Independent Publishing Platform, USA (2015)
41. Phan, D., Yang, J., Grosu, R., Smolka, S.A., Stoller, S.D.: Collision avoidance for mobile robots with limited sensing and limited information about moving obstacles. Formal Methods Syst. Des. **51**(1), 62–86 (2017)
42. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, 11–13 January 1989, pp. 179–190 (1989)
43. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons Inc., New York (2005)
44. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete-event processes. SIAM J. Control Optim. **25**(1), 206–230 (1987)
45. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1), 30–50 (2000)
46. Seshia, S. A., Sadigh, D.: Towards verified artificial intelligence. CoRR, arXiv:1606.08514 (2016)
47. Sha, L.: Using simplicity to control complexity. IEEE Soft. **18**(4), 20–28 (2001)
48. Velner, Y., Chatterjee, K., Doyen, L., Henzinger, T.A., Rabinovich, A.M., Raskin, J.-F.: The complexity of multi-mean-payoff and multi-energy games. Inf. Comput. **241**, 177–196 (2015)
49. Wu, Y., Raman, V., Rawlings, B.C., Lafortune, S., Seshia, S.A.: Synthesis of obfuscation policies to ensure privacy and utility. J. Autom. Reasoning **60**(1), 107–131 (2018)

Taming Delays in Dynamical Systems

Unbounded Verification of Delay Differential Equations

Shenghua Feng[1,2], Mingshuai Chen[1,2(✉)], Naijun Zhan[1,2(✉)], Martin Fränzle[3], and Bai Xue[1,2]

[1] SKLCS, Institute of Software, CAS, Beijing, China
{fengsh,chenms,znj,xuebai}@ios.ac.cn
[2] University of Chinese Academy of Sciences, Beijing, China
[3] Carl von Ossietzky Universität Oldenburg, Oldenburg, Germany
fraenzle@informatik.uni-oldenburg.de

Abstract. Delayed coupling between state variables occurs regularly in technical dynamical systems, especially embedded control. As it consequently is omnipresent in safety-critical domains, there is an increasing interest in the safety verification of systems modelled by Delay Differential Equations (DDEs). In this paper, we leverage qualitative guarantees for the existence of an exponentially decreasing estimation on the solutions to DDEs as established in classical stability theory, and present a quantitative method for constructing such delay-dependent estimations, thereby facilitating a reduction of the verification problem over an unbounded temporal horizon to a bounded one. Our technique builds on the linearization technique of nonlinear dynamics and spectral analysis of the linearized counterparts. We show experimentally on a set of representative benchmarks from the literature that our technique indeed extends the scope of bounded verification techniques to unbounded verification tasks. Moreover, our technique is easy to implement and can be combined with any automatic tool dedicated to bounded verification of DDEs.

Keywords: Unbounded verification · Delay Differential Equations (DDEs) · Safety and stability · Linearization · Spectral analysis

1 Introduction

The theory of dynamical systems featuring delayed coupling between state variables dates back to the 1920s, when Volterra [41,42], in his research on predator-prey models and viscoelasticity, formulated some rather general differential equations incorporating the past states of the system. This formulation, now known as delay differential equations (DDEs), was developed further by, e.g., Mishkis [30] and Bellman

This work has been supported through grants by NSFC under grant No. 61625206, 61732001 and 61872341, by Deutsche Forschungsgemeinschaft through grants No. GRK 1765 and FR 2715/4, and by the CAS Pioneer Hundred Talents Program under grant No. Y8YC235015.

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 650–669, 2019.
https://doi.org/10.1007/978-3-030-25540-4_37

and Cooke [2], and has witnessed numerous applications in many domains. Prominent examples include population dynamics [25], where birth rate follows changes in population size with a delay related to reproductive age; spreading of infectious diseases [5], where delay is induced by the incubation period; or networked control systems [21] with their associated transport delays when forwarding data through the communication network. These applications range further to models in optics [23], economics [38], and ecology [13], to name just a few. Albeit resulting in more accurate models, the presence of time delays in feedback dynamics often induces considerable extra complexity when one attempts to design or even verify such dynamical systems. This stems from the fact that the presence of feedback delays reduces controllability due to the impossibility of immediate reaction and enhances the likelihood of transient overshoot or even oscillation in the feedback system, thus violating safety or stability certificates obtained on idealized, delay-free models of systems prone to delayed coupling.

Though established automated methods addressing ordinary differential equations (ODEs) and their derived models, like hybrid automata, have been extensively studied in the verification literature, techniques pertaining to ODEs do not generalize straightforwardly to delayed dynamical systems described by DDEs. The reason is that the future evolution of a DDE is no longer governed by the current state instant only, but depends on a chunk of its historical trajectory, such that introducing even a single constant delay immediately renders a system with finite-dimensional states into an infinite-dimensional dynamical system. There are approximation methods, say the Padé approximation [39], that approximate DDEs with finite-dimensional models, which however may hide fundamental behaviors, e.g. (in-)stability, of the original delayed dynamics, as remarked in Sect. 5.2.2.8.1 of [26]. Consequently, despite well-developed numerical methods for solving DDEs as well as methods for stability analysis in the realm of control theory, hitherto in automatic verification, only a few approaches address the effects of delays due to the immediate impact of delays on the structure of the state spaces to be traversed by state-exploratory methods.

In this paper, we present a constructive approach dedicated to verifying safety properties of delayed dynamical systems encoded by DDEs, where the safety properties pertain to an infinite time domain. This problem is of particular interests when one pursues correctness guarantees concerning dynamics of safety-critical systems over a long run. Our approach builds on the *linearization* technique of potentially nonlinear dynamics and *spectral analysis* of the linearized counterparts. We leverage qualitative guarantees for the existence of an exponentially decreasing estimation on the solutions to DDEs as established in classical stability theory (see, e.g., [2,19,24]), and present a quantitative method to construct such estimations, thereby reducing the temporally unbounded verification problems to their bounded counterparts.

The class of systems we consider features delayed differential dynamics governed by DDEs of the form $\dot{\mathbf{x}}(t) = \boldsymbol{f}(\mathbf{x}(t), \mathbf{x}(t-r_1), \ldots, \mathbf{x}(t-r_k))$ with initial states specified by a continuous function $\boldsymbol{\phi}(t)$ on $[-r_{\max}, 0]$ where $r_{\max} = \max\{r_1, \ldots, r_k\}$. It thus involves a combination of ODE and DDE with multiple constant delays $r_i > 0$, and has been successfully used to model various real-world systems in the aforementioned fields. In general, formal verification of unbounded safety or, dually, reachability properties of such systems inherits undecidability from similar properties for ODEs

(cf. e.g., [14]). We therefore tackle this unbounded verification problem by leveraging a stability criterion of the system under investigation.

Contributions. In this paper, we present a quantitative method for constructing a delay-dependent, exponentially decreasing upper bound, if existent, that encloses trajectories of a DDE originating from a certain set of initial functions. This method consequently yields a temporal bound T^* such that for any $T > T^*$, the system is safe over $[-r_{\max}, T]$ iff it is safe over $[-r_{\max}, \infty)$. For linear dynamics, such an equivalence of safety applies to any initial set of functions drawn from a compact subspace in $\mathbb{R}^n$; while for nonlinear dynamics, our approach produces (a subset of) the *basin of attraction* around a *steady state*, and therefore a certificate (by bounded verification in finitely many steps) that guarantees the reachable set being contained in this basin suffices to claim safety/unsafety of the system over an infinite time horizon. Our technique is easy to implement and can be combined with any automatic tool for bounded verification of DDEs. We show experimentally on a set of representative benchmarks from the literature that our technique effectively extends the scope of bounded verification techniques to unbounded verification tasks.

Related Work. As surveyed in [14], the research community has over the past three decades vividly addressed automatic verification of hybrid discrete-continuous systems in a safety-critical context. The almost universal undecidability of the unbounded reachability problem, however, confines the sound key-press routines to either semi-decision procedures or even approximation schemes, most of which address bounded verification by computing the finite-time image of a set of initial states. It should be obvious that the functional rather than state-based nature of the initial condition of DDEs prevents a straightforward generalization of this approach.

Prompted by actual engineering problems, the interest in safety verification of continuous or hybrid systems featuring delayed coupling is increasing recently. We classify these contributions into two tracks. The first track pursues propagation-based bounded verification: Huang et al. presented in [21] a technique for simulation-based time-bounded invariant verification of nonlinear networked dynamical systems with delayed interconnections, by computing bounds on the sensitivity of trajectories to changes in initial states and inputs of the system. A method adopting the paradigm of verification-by-simulation (see, e.g., [9,16,31]) was proposed in [4], which integrates rigorous error analysis of the numeric solving and the sensitivity-related state bloating algorithms (cf. [7]) to obtain safe enclosures of time-bounded reachable sets for systems modelled by DDEs. In [46], the authors identified a class of DDEs featuring a local homeomorphism property which facilitates construction of over- and under-approximations of reachable sets by performing reachability analysis on the boundaries of the initial sets. Goubault et al. presented in [17] a scheme to compute inner- and outer-approximating flowpipes for DDEs with uncertain initial states and parameters using Taylor models combined with space abstraction in the shape of zonotopes. The other track of the literature tackles unbounded reachability problem of DDEs by taking into account the asymptotic behavior of the dynamics under investigation, captured by, e.g., Lyapunov functions in [32,47] and barrier certificates in [35]. These approaches however share a common limitation that a polynomial template has to be specified either for the interval

Taylor models exploited in [47] (and its extension [29] to cater for properties specified as bounded metric interval temporal logic (MITL) formulae), for Lyapunov functionals in [32], or for barrier certificates in [35]. Our approach drops this limitation by resorting to the linearization technique followed by spectral analysis of the linearized counterparts, and furthermore extends over [47] by allowing immediate feedback (i.e. $\mathbf{x}(t)$) as well as multiple delays in the dynamics), to which their technique does not generalize immediately. In contrast to the *absolute stability* exploited in [32], namely a criterion that ensures stability for arbitrarily large delays, we give the construction of a delay-dependent stability certificate thereby substantially increasing the scope of dynamics amenable to stability criteria, for instance, the famous Wright's equation (cf. [44]). Finally, we refer the readers to [34] and [33] for related contributions in showing the existence of abstract symbolic models for nonlinear control systems with time-varying and unknown time-delay signals via approximate bisimulations.

2 Problem Formulation

Notations. Let $\mathbb{N}$, $\mathbb{R}$ and $\mathbb{C}$ be the set of natural, real and complex numbers, respectively. Vectors will be denoted by boldface letters. For $z = a + \mathrm{i}b \in \mathbb{C}$ with $a, b \in \mathbb{R}$, the real and imaginary parts of z are denoted respectively by $\Re(z) = a$ and $\Im(z) = b$; $|z| = \sqrt{a^2 + b^2}$ is the modulus of z. For a vector $\mathbf{x} \in \mathbb{R}^n$, x_i refers to its i-th component, and its maximum norm is denoted by $\|\mathbf{x}\| = \max_{1 \le i \le n} |x_i|$. We define for $\delta > 0$, $\mathcal{B}(\mathbf{x}, \delta) = \{\mathbf{x}' \in \mathbb{R}^n \mid \|\mathbf{x}' - \mathbf{x}\| \le \delta\}$ as the δ-closed ball around $\mathbf{x}$. The notation $\|\cdot\|$ extends to a set $X \subseteq \mathbb{R}^n$ as $\|X\| = \sup_{\mathbf{x} \in X} \|\mathbf{x}\|$, and to an $m \times n$ complex-valued matrix A as $\|A\| = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|$. $\overline{X}$ is the closure of X and ∂X denotes the boundary of X. For $a \le b$, let $C^0([a, b], \mathbb{R}^n)$ denote the space of continuous functions from $[a, b]$ to $\mathbb{R}^n$, which is associated with the maximum norm $\|f\| = \max_{t \in [a,b]} \|f(t)\|$. We abbreviate $C^0([-r, 0], \mathbb{R}^n)$ as $\mathcal{C}_r$ for a fixed positive constant r, and let C^1 consist of all continuously differentiable functions. Given $f \colon [0, \infty) \mapsto \mathbb{R}$ a measurable function such that $\|f(t)\| \le a\mathrm{e}^{bt}$ for some constants a and b, then the Laplace transform $\mathcal{L}\{f\}$ defined by $\mathcal{L}\{f\}(z) = \int_0^\infty \mathrm{e}^{-zt} f(t)\,\mathrm{d}t$ exists and is an analytic function of z for $\Re(z) > b$.

Delayed Differential Dynamics. We consider a class of dynamical systems featuring delayed differential dynamics governed by DDEs of autonomous type:

$$\begin{cases} \dot{\mathbf{x}}(t) = \boldsymbol{f}(\mathbf{x}(t), \mathbf{x}(t - r_1), \ldots, \mathbf{x}(t - r_k)), & t \in [0, \infty) \\ \mathbf{x}(t) = \boldsymbol{\phi}(t), & t \in [-r_k, 0] \end{cases} \tag{1}$$

where $\mathbf{x}$ is the time-dependent *state* vector in $\mathbb{R}^n$, $\dot{\mathbf{x}}$ denotes its temporal derivative $\mathrm{d}\mathbf{x}/\mathrm{d}t$, and t is a real variable modelling time. The discrete delays are assumed to be ordered as $r_k > \ldots > r_1 > 0$, and the initial states are specified by a vector-valued function $\boldsymbol{\phi} \in \mathcal{C}_{r_k}$.

Suppose $\boldsymbol{f}$ is a Lipschitz-continuous vector-valued function in $C^1\left(\mathbb{R}^{(k+1)n}, \mathbb{R}^n\right)$, which implies that the system has a unique maximal *solution* (or *trajectory*) from a given initial condition $\boldsymbol{\phi} \in \mathcal{C}_{r_k}$, denoted as $\boldsymbol{\xi}_{\boldsymbol{\phi}} \colon [-r_k, \infty) \mapsto \mathbb{R}^n$. We denote in the

sequel by $\boldsymbol{f}_{\mathbf{x}} \widehat{=} \left[\frac{\partial \boldsymbol{f}}{\partial x_1} \cdots \frac{\partial \boldsymbol{f}}{\partial x_n}\right]$ the Jacobian matrix (i.e., matrix consisting of all first-order partial derivatives) of $\boldsymbol{f}$ w.r.t. the component $\mathbf{x}(t)$. Similar notations apply to components $\mathbf{x}(t - r_i)$, for $i = 1, \ldots, k$.

Example 1 (Gene regulation [12,36]*).* The control of gene expression in cells is often modelled with time delays in equations of the form

$$\begin{cases} \dot{x_1}(t) = g\left(x_n(t - r_n)\right) - \beta_1 x_1(t) \\ \dot{x_j}(t) = x_{j-1}(t - r_{j-1}) - \beta_j x_j(t), \quad 1 < j \leq n \end{cases} \tag{2}$$

where the gene is transcribed producing mRNA (x_1), which is translated into enzyme x_2 that in turn produces another enzyme x_3 and so on. The end product x_n acts to repress the transcription of the gene by $\dot{g} < 0$. Time delays are introduced to account for time involved in transcription, translation, and transport. The positive β_j's represent decay rates of the species. The dynamic described in Eq. (2) falls exactly into the scope of systems considered in this paper, and in fact, it instantiates a more general family of systems known as monotone cyclic feedback systems (MCFS) [28], which includes neural networks, testosterone control, and many other effects in systems biology.

Lyapunov Stability. Given a system of DDEs in Eq. (1), suppose $\boldsymbol{f}$ has a steady state (a.k.a., *equilibrium*) at $\mathbf{x}_e$ such that $\boldsymbol{f}(\mathbf{x}_e, \ldots, \mathbf{x}_e) = \mathbf{0}$ then

- $\mathbf{x}_e$ is said to be *Lyapunov stable*, if for every $\epsilon > 0$, there exists $\delta > 0$ such that, if $\|\boldsymbol{\phi} - \mathbf{x}_e\| < \delta$, then for every $t \geq 0$ we have $\|\boldsymbol{\xi}_{\boldsymbol{\phi}}(t) - \mathbf{x}_e\| < \epsilon$.
- $\mathbf{x}_e$ is said to be *asymptotically stable*, if it is Lyapunov stable and there exists $\delta > 0$ such that, if $\|\boldsymbol{\phi} - \mathbf{x}_e\| < \delta$, then $\lim_{t \to \infty} \|\boldsymbol{\xi}_{\boldsymbol{\phi}}(t) - \mathbf{x}_e\| = 0$.
- $\mathbf{x}_e$ is said to be *exponentially stable*, if it is asymptotically stable and there exist $\alpha, \beta, \delta > 0$ such that, if $\|\boldsymbol{\phi} - \mathbf{x}_e\| < \delta$, then $\|\boldsymbol{\xi}_{\boldsymbol{\phi}}(t) - \mathbf{x}_e\| \leq \alpha \|\boldsymbol{\phi} - \mathbf{x}_e\| \, \mathrm{e}^{-\beta t}$, for all $t \geq 0$. The constant β is called the *rate of convergence*.

Here $\mathbf{x}_e$ can be generalized to a constant function in $\mathcal{C}_{r_k}$ when employing the supremum norm $\|\boldsymbol{\phi} - \mathbf{x}_e\|$ over functions. This norm further yields the *locality* of the above definitions, i.e., they describe the behavior of a system near an equilibrium, rather than of all initial conditions $\boldsymbol{\phi} \in \mathcal{C}_{r_k}$, in which case it is termed the *global stability*. W.l.o.g., we assume $\boldsymbol{f}(\mathbf{0}, \ldots, \mathbf{0}) = \mathbf{0}$ in the sequel and investigate the stability of the zero equilibrium thereof. Any nonzero equilibrium can be straightforwardly shifted to a zero one by coordinate transformation while preserving the stability properties, see e.g., [19].

Safety Verification Problem. Given $\mathcal{X} \subseteq \mathbb{R}^n$ a compact set of initial states and $\mathcal{U} \subseteq \mathbb{R}^n$ a set of unsafe or otherwise bad states, a delayed dynamical system of the form (1) is said to be T-*safe* iff all trajectories originating from any $\boldsymbol{\phi}(t)$ satisfying $\boldsymbol{\phi}(t) \in \mathcal{X}, \forall t \in [-r_k, 0]$ do not intersect with $\mathcal{U}$ at any $t \in [-r_k, T]$, and T-*unsafe* otherwise. In particular, we distinguish *unbounded verification* with $T = \infty$ from *bounded verification* with $T < \infty$.

In subsequent sections, we first present our approach to tackling the safety verification problem of delayed differential dynamics coupled with one single constant delay (i.e., $k = 1$ in Eq. (1)) in an unbounded time domain, by leveraging a quantitative

stability criterion, if existent, for the linearized counterpart of the potentially nonlinear dynamics in question. A natural extension of this approach to cater for dynamics with multiple delay terms will be remarked thereafter. In what follows, we start the elaboration of the method from DDEs of linear dynamics that admit spectral analysis, and move to nonlinear cases afterwards and show how the linearization technique can be exploited therein.

3 Linear Dynamics

Consider the linear sub-class of dynamics given in Eq. (1):

$$\begin{cases} \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{x}(t-r), & t \in [0,\infty) \\ \mathbf{x}(t) = \boldsymbol{\phi}(t), & t \in [-r,0] \end{cases} \tag{3}$$

where $A, B \in \mathbb{R}^{n\times n}$, $\boldsymbol{\phi} \in \mathcal{C}_r$, and the system is associated with the *characteristic equation*

$$\det\left(zI - A - B\mathrm{e}^{-rz}\right) = 0, \tag{4}$$

where I is the $n\times n$ identity matrix. Denote by $h(z) \mathrel{\widehat{=}} zI - A - B\mathrm{e}^{-rz}$ the *characteristic matrix* in the sequel. Notice that the characteristic equation can be obtained by seeking nontrivial solutions to Eq. (3) of the form $\boldsymbol{\xi}_\phi(t) = \mathbf{c}\mathrm{e}^{zt}$, where $\mathbf{c}$ is an n-dimensional nonzero constant vector.

The roots $\lambda \in \mathbb{C}$ of Eq. (4) are called *characteristic roots* or *eigenvalues* and the set of all eigenvalues is referred to as the *spectrum*, denoted by $\sigma = \{\lambda \mid \det(h(\lambda)) = 0\}$. Due to the exponentiation in the characteristic equation, the DDE has, in line with its infinite-dimensional nature, infinitely many eigenvalues possibly, making a spectral analysis more involved. The spectrum does however enjoy some elementary properties that can be exploited in the analysis. For instance, the spectrum has no finite accumulation point in $\mathbb{C}$ and therefore for each positive $\gamma \in \mathbb{R}$, the number of roots satisfying $|\lambda| \le \gamma$ is finite. It follows that the spectrum is a countable (albeit possibly infinite) set:

Lemma 1 (Accumulation freedom [6,19]). *Given $\gamma \in \mathbb{R}$, there are at most finitely many characteristic roots satisfying $\Re(\lambda) > \gamma$. If there is a sequence $\{\lambda_n\}$ of roots of Eq. (4) such that $|\lambda_n| \to \infty$ as $n \to \infty$, then $\Re(\lambda_n) \to -\infty$ as $n \to \infty$.*

Lemma 1 suggests that there are only a finite number of solutions in any vertical strip in the complex plane, and there thus exists an upper bound $\alpha \in \mathbb{R}$ such that every characteristic root λ in the spectrum satisfies $\Re(\lambda) < \alpha$. This upper bound captures essentially the asymptotic behavior of the linear dynamics:

Theorem 1 (Globally exponential stability [6,36]). *Suppose $\Re(\lambda) < \alpha$ for every characteristic root λ. Then there exists $K > 0$ such that*

$$\|\boldsymbol{\xi}_\phi(t)\| \le K\|\boldsymbol{\phi}\|\mathrm{e}^{\alpha t}, \quad \forall t \ge 0,\ \forall \boldsymbol{\phi} \in \mathcal{C}_r, \tag{5}$$

where $\boldsymbol{\xi}_\phi(t)$ is the solution to Eq. (3). In particular, $\mathbf{x} = \mathbf{0}$ is a globally exponentially stable equilibrium of Eq. (3) if $\Re(\lambda) < 0$ for every characteristic root; it is unstable if there is a root satisfying $\Re(\lambda) > 0$.

Theorem 1 establishes an existential guarantee that the solution to the linear delayed dynamics approaches the zero equilibrium exponentially for any initial conditions in $\mathcal{C}_r$. To achieve automatic safety verification, however, we ought to find a constructive means of estimating the (signed) rate of convergence α and the coefficient K in Eq. (5). This motivates the introduction of the so-called *fundamental solution* $\boldsymbol{\xi}_{\phi'}(t)$ to Eq. (3), whose Laplace transform will later be shown to be $h^{-1}(z)$, the inverse characteristic matrix, which always exists for z satisfying $\Re(z) > \max_{\lambda\in\sigma} \Re(\lambda)$.

Lemma 2 (Variation-of-constants [19,36]). *Let $\boldsymbol{\xi}_{\phi}(t)$ be the solution to Eq. (3). Denote by $\boldsymbol{\xi}_{\phi'}(t)$ the solution that satisfies Eq. (3) for $t \geq 0$ and satisfies a variation of the initial condition as $\boldsymbol{\phi}'(0) = I$ and $\boldsymbol{\phi}'(t) = O$ for all $t \in [-r, 0)$, where O is the $n \times n$ zero matrix, then for $t \geq 0$,*

$$\boldsymbol{\xi}_{\phi}(t) = \boldsymbol{\xi}_{\phi'}(t)\boldsymbol{\phi}(0) + \int_0^t \boldsymbol{\xi}_{\phi'}(t-\tau) B \boldsymbol{\phi}(\tau - r)\, \mathrm{d}\tau. \tag{6}$$

Note that in Eq. (6), $\boldsymbol{\phi}(t)$ is extended to $[-r, \infty)$ by making it zero for $t > 0$. In spite of the discontinuity of $\boldsymbol{\phi}'$ at zero, the existence of the solution $\boldsymbol{\xi}_{\phi'}(t)$ can be proven by the well-known method of steps [8].

Lemma 3 (Fundamental solution [19]). *The solution $\boldsymbol{\xi}_{\phi'}(t)$ to Eq. (3) with initial data $\boldsymbol{\phi}'$ is the fundamental solution; that is for z s.t. $\Re(z) > \max_{\lambda\in\sigma} \Re(\lambda)$,*

$$\mathcal{L}\{\boldsymbol{\xi}_{\phi'}\}(z) = h^{-1}(z).$$

The fundamental solution $\boldsymbol{\xi}_{\phi'}(t)$ can be proven to share the same exponential bound as that in Theorem 1, while the following theorem, as a consequence of Lemma 2, gives an exponential estimation of $\boldsymbol{\xi}_{\phi}(t)$ in connection with $\boldsymbol{\xi}_{\phi'}(t)$:

Theorem 2 (Exponential estimation [36]). *Denote by $\mu \mathrel{\widehat{=}} \max_{\lambda\in\sigma} \Re(\lambda)$ the maximum real part of eigenvalues in the spectrum. Then for any $\alpha > \mu$, there exists $K > 0$ such that*

$$\|\boldsymbol{\xi}_{\phi'}(t)\| \leq K \mathrm{e}^{\alpha t}, \quad \forall t \geq 0, \tag{7}$$

and hence by Eq. (6), $\|\boldsymbol{\xi}_{\phi}(t)\| \leq K\left(1 + \|B\| \int_0^r \mathrm{e}^{-\alpha\tau}\, \mathrm{d}\tau\right) \|\boldsymbol{\phi}\|\, \mathrm{e}^{\alpha t}$ for any $t \geq 0$ and $\boldsymbol{\phi} \in \mathcal{C}_r$. In particular, $\mathbf{x} = \mathbf{0}$ is globally exponentially stable for Eq. (3) if $\mu < 0$.

Following Theorem 2, an exponentially decreasing bound on the solution $\boldsymbol{\xi}_{\phi}(t)$ to linear DDEs of the form (3) can be assembled by computing α satisfying $\mu < \alpha < 0$ and the coefficient $K > 0$.

3.1 Identifying the Rightmost Roots

Due to the significance of characteristic roots in the context of stability and bifurcation analysis, numerical methods on identifying—particularly the rightmost—roots of linear (or linearized) DDEs have been extensively studied in the past few decades, see e.g., [3,11,43,45]. There are indeed complete methods on isolating real roots of polynomial exponential functions, for instances [37] and [15] based on cylindrical algebraic decomposition (CAD). Nevertheless, as soon as non-trivial exponential functions arise

in the characteristic equation, there appear to be few, if any, symbolic approaches to detecting complex roots of the equation.

In this paper, we find α that bounds the spectrum from the right of the complex plane, by resorting to the numerical approach developed in [11]. The computation therein employs discretization of the solution operator using linear multistep (LMS) methods to approximate eigenvalues of linear DDEs with multiple constant delays, under an absolute error of $\mathcal{O}(\tau^p)$ for sufficiently small stepsize τ, where $\mathcal{O}(\cdot)$ is the big Omicron notation and p depends on the order of the LMS-methods. A well-developed MATLAB package called DDE-BIFTOOL [10] is furthermore available to mechanize the computation, which will be demonstrated in our forthcoming examples.

3.2 Constructing K

By the inverse Laplace transform (cf. Theorem 5.2 in [19] for a detailed proof), we have $\boldsymbol{\xi}_{\phi'}(t) = \lim_{V\to\infty} \frac{1}{2\pi \mathrm{i}} \int_{\alpha-\mathrm{i}V}^{\alpha+\mathrm{i}V} \mathrm{e}^{zt} h^{-1}(z)\,\mathrm{d}z$ for z satisfying $\Re(z) > \mu$, where α is the exponent associated with the bound on $\boldsymbol{\xi}_{\phi'}(t)$ in Eq. (7), and hence by substituting $z = \alpha + \mathrm{i}\nu$, we have

$$\mathrm{e}^{-\alpha t}\boldsymbol{\xi}_{\phi'}(t) = \lim_{V\to\infty} \frac{1}{2\pi} \int_{-V}^{V} \mathrm{e}^{\mathrm{i}\nu t} h^{-1}(\alpha + \mathrm{i}\nu)\,\mathrm{d}\nu.$$

Since $h^{-1}(z) = \frac{I}{z} + \left(h^{-1}(z) - \frac{I}{z}\right) = \frac{I}{z} + \mathcal{O}\left(1/z^2\right)$, together with the fact that an integral over a quadratic integrand is convergent, it follows that

$$\mathrm{e}^{-\alpha t}\boldsymbol{\xi}_{\phi'}(t) = \lim_{V\to\infty} \frac{1}{2\pi} \int_{-V}^{V} \mathrm{e}^{\mathrm{i}\nu t} \frac{I}{\alpha + \mathrm{i}\nu}\,\mathrm{d}\nu + \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathrm{e}^{\mathrm{i}\nu t} \mathcal{O}\left(\frac{1}{(\alpha + \mathrm{i}\nu)^2}\right) \mathrm{d}\nu.$$

By taking the norm while observing that $\left|\mathrm{e}^{\mathrm{i}\nu t}\right| = 1$, we get

$$\mathrm{e}^{-\alpha t} \left\|\boldsymbol{\xi}_{\phi'}(t)\right\| \le \Bigg\| \lim_{V\to\infty} \frac{1}{2\pi} \underbrace{\int_{-V}^{V} \mathrm{e}^{\mathrm{i}\nu t} \frac{I}{\alpha + \mathrm{i}\nu}\,\mathrm{d}\nu}_{\text{(8-a)}} \Bigg\| + \frac{1}{2\pi} \underbrace{\int_{-\infty}^{\infty} \left\| \mathcal{O}\left(\frac{1}{(\alpha + \mathrm{i}\nu)^2}\right) \right\| \mathrm{d}\nu}_{\text{(8-b)}}. \tag{8}$$

For the integral (8-a), the fact[1] that

$$\int_{-\infty}^{\infty} \frac{\mathrm{e}^{\mathrm{i}ax}}{b + \mathrm{i}x}\,\mathrm{d}x = \int_{-\infty}^{\infty} \frac{\mathrm{e}^{\mathrm{i}x}}{ab + \mathrm{i}x}\,\mathrm{d}x = \begin{cases} 2\pi \mathrm{e}^{-ab} & \text{if } a, b > 0 \\ 0 & \text{if } a > 0, b < 0, \end{cases} \tag{9}$$

implies

$$\left\| \lim_{V\to\infty} \frac{1}{2\pi} \int_{-V}^{V} \mathrm{e}^{\mathrm{i}\nu t} \frac{I}{\alpha + \mathrm{i}\nu}\,\mathrm{d}\nu \right\| \le \begin{cases} 1, & \forall t > 0,\ \forall \alpha > 0 \\ 0, & \forall t > 0,\ \forall \alpha < 0. \end{cases} \tag{10}$$

Notice that the second integral (8-b) is computable, since it is convergent and independent of t. The underlying computation of the *improper integral*, however, can be rather time-consuming. We therefore detour by computing an upper bound of (8-b) in the form of a *definite integral*, due to Lemma 4, which suffices to constitute an exponential estimation of $\boldsymbol{\xi}_{\phi'}(t)$ while reducing computational efforts pertinent to the integration.

[1] The integral in (9) is divergent for $a = 0$ or $b = 0$ in the sense of a Riemann integral.

Lemma 4. *There exists $M > 0$ such that inequation (11) below holds for any $\alpha > \mu$.*

$$\int_{-\infty}^{\infty} \left\| \mathcal{O}\left(\frac{1}{(\alpha+\mathrm{i}\nu)^2}\right) \right\| \mathrm{d}\nu \le \int_{-M}^{M} \left\| \mathcal{O}\left(\frac{1}{(\alpha+\mathrm{i}\nu)^2}\right) \right\| \mathrm{d}\nu + \frac{8n}{M}\left(\|A\| + \|B\|\,\mathrm{e}^{-r\alpha}\right) \tag{11}$$

where $\mu \widehat{=} \max_{\lambda\in\sigma} \Re(\lambda)$, $z = \alpha + \mathrm{i}\nu$, and n is the order of A and B.

Proof. The proof depends essentially on constructing a threshold $M > 0$ such that the integral over $|\nu| > M$ can be bounded, thus transforming the improper integral in question to a definite one. To find such an M, observe that

$$\left\| \mathcal{O}\left(\frac{1}{z^2}\right) \right\| = \left\| h^{-1}(z) - \frac{I}{z} \right\| = \left\| h^{-1}(z) \right\| \left\| I - \frac{h(z)}{z} \right\| \le \frac{\left\| h^{-1}(z) \right\|}{|z|}(\|A\| + \|B\|\,\mathrm{e}^{-r\alpha}).$$

Without loss of generality, suppose the entry of $h^{-1}(z)$ at (i, j) takes the form

$$\begin{aligned}\left(h^{-1}\right)_{ij} &= \left(\sum_{k=0}^{n-1} p_k^{ij}(\mathrm{e}^{-rz})z^k\right)/\det(h(z)) = \left(\sum_{k=0}^{n-1} p_k^{ij}(\mathrm{e}^{-rz})z^k\right)/\left(z^n + \sum_{k=0}^{n-1} q_k(\mathrm{e}^{-rz})z^k\right) \\ &= \frac{1}{z}\left(\sum_{k=0}^{n-1} p_k^{ij}(\mathrm{e}^{-rz})z^{k-n+1}\right)/\left(1 + \sum_{k=0}^{n-1} q_k(\mathrm{e}^{-rz})z^{k-n}\right),\end{aligned}$$

where $p_k^{ij}(\cdot)$ and $q_k(\cdot)$ are polynomials in e^{-rz} as coefficients of z^k. Since e^{-rz} is bounded by $\mathrm{e}^{-r\alpha}$ along the vertical line $z = \alpha + \mathrm{i}\nu$, we can conclude that there exist P_k^{ij} and Q_k such that $\left|p_k^{ij}(\mathrm{e}^{-rz})\right| \le P_k^{ij}$ and $|q_k(\mathrm{e}^{-rz})| \le Q_k$, with $P_{n-1}^{ij} = 1$ if $i = j$, and 0 otherwise. Furthermore, in the vertical line $z = \alpha + \mathrm{i}\nu$, if $|\nu| \ge 1$, then

$$\begin{aligned}\left|\sum_{k=0}^{n-1} p_k^{ij}(\mathrm{e}^{-rz})z^{k-n+1}\right| &\le \left|p_{n-1}^{ij}(\mathrm{e}^{-rz})\right| + \sum_{k=0}^{n-2}\left|p_k^{ij}(\mathrm{e}^{-rz})z^{-1}\right| \le P_{n-1}^{ij} + \sum_{k=0}^{n-2} P_k^{ij}\left|z^{-1}\right|, \\ \left|1 + \sum_{k=0}^{n-1} q_k(\mathrm{e}^{-rz})z^{k-n}\right| &\ge 1 - \sum_{k=0}^{n-1}\left|q_k(\mathrm{e}^{-rz})\right|\left|z^{k-n}\right| \ge 1 - \sum_{k=0}^{n-1} Q_k\left|z^{-1}\right|.\end{aligned}$$

We can thus choose $|\nu| > M \widehat{=} \max\limits_{1\le i,j\le n}\left\{1, 2\sum\limits_{k=0}^{n-1} Q_k, \sum\limits_{k=0}^{n-2} P_k^{ij}\right\}$, which implies

$$\begin{aligned}\left|\left(\sum_{k=0}^{n-1} p_k^{ij}(\mathrm{e}^{-rz})z^k\right)/\det(h(z))\right| &\le \left|\frac{1}{z}\left(\sum_{k=0}^{n-1} p_k^{ij}(\mathrm{e}^{-rz})z^{k-n+1}\right)/\left(1 + \sum_{k=0}^{n-1} q_k(\mathrm{e}^{-rz})z^{k-n}\right)\right| \\ &\le \left|\frac{1}{z}\right|\left(P_{n-1}^{ij} + \sum_{k=0}^{n-2} P_k^{ij}\left|z^{-1}\right|\right)/\left(1 - \sum_{k=0}^{n-1} Q_k\left|z^{-1}\right|\right) \le \frac{2}{|z|}(1 + P_{n-1}^{ij}) \le \frac{4}{|z|},\end{aligned}$$

where the third inequality holds since $|\nu| > M$. It then follows, if $|\nu| > M$, that

$$\left\| \mathcal{O}\left(\frac{1}{(\alpha+\mathrm{i}\nu)^2}\right) \right\| \le \frac{\left\| h^{-1}(z) \right\|}{|z|}(\|A\| + \|B\|\,\mathrm{e}^{-r\alpha}) \le \frac{4n}{\nu^2}(\|A\| + \|B\|\,\mathrm{e}^{-r\alpha}),$$

and thereby

$$\int_{-\infty}^{\infty} \left\| \mathcal{O}\left(\frac{1}{(\alpha+\mathrm{i}\nu)^2}\right)\right\| \leq \int_{-M}^{M} \left\| \mathcal{O}\left(\frac{1}{(\alpha+\mathrm{i}\nu)^2}\right)\right\| \mathrm{d}\nu + 2\int_{M}^{\infty} \frac{4n}{\nu^2}(\|A\| + \|B\|\,\mathrm{e}^{-r\alpha})\,\mathrm{d}\nu$$
$$\leq \int_{-M}^{M} \left\| \mathcal{O}\left(\frac{1}{(\alpha+\mathrm{i}\nu)^2}\right)\right\| \mathrm{d}\nu + \frac{8n}{M}\left(\|A\| + \|B\|\,\mathrm{e}^{-r\alpha}\right).$$

This completes the proof. □

Equations (8), (10) and (11) yield that $\mathrm{e}^{-\alpha t}\,\|\boldsymbol{\xi}_{\phi'}(t)\|$ is upper-bounded by

$$K = \frac{1}{2\pi}\left(\int_{-M}^{M} \left\| \mathcal{O}\left(\frac{1}{(\alpha+\mathrm{i}\nu)^2}\right)\right\| \mathrm{d}\nu + \frac{8n}{M}\left(\|A\| + \|B\|\,\mathrm{e}^{-r\alpha}\right)\right) + 1_0(\alpha), \quad (12)$$

for all $t > 0$. Here M is the constant given in Lemma 4, while $1_0\colon (\mu,\infty)\setminus\{0\} \mapsto \{0,1\}$ is an indicator function[2] of $\{\alpha \mid \alpha > 0\}$, i.e., $1_0(\alpha) = 1$ for $\alpha > 0$ and $1_0(\alpha) = 0$ for $\mu < \alpha < 0$.

In contrast to the existential estimation guarantee established in Theorem 2, exploiting the construction of α and K gives a constructive quantitative criterion permitting to reduce an unbounded safety verification problem to its bounded counterpart:

Theorem 3 (Equivalence of bounded and unbounded safety). *Given $\mathcal{X} \subseteq \mathbb{R}^n$ a set of initial states and $\mathcal{U} \subseteq \mathbb{R}^n$ a set of bad states satisfying $\mathbf{0} \notin \overline{\mathcal{U}}$, suppose we have α satisfying $\mu < \alpha < 0$ and K from Eq. (12). Let $\hat{K} \mathrel{\widehat{=}} K\left(1 + \|B\|\int_0^r \mathrm{e}^{-\alpha\tau}\,\mathrm{d}\tau\right)\|\mathcal{X}\|$, then there exists $T^* < \infty$, defined as*

$$T^* \mathrel{\widehat{=}} \max\{0, \inf\{T \mid \forall t > T\colon [-\hat{K}\mathrm{e}^{\alpha t}, \hat{K}\mathrm{e}^{\alpha t}]^n \cap \mathcal{U} = \emptyset\}\}, \quad (13)$$

such that for any $T > T^*$, the system (3) is ∞-safe iff it is T-safe.

Proof. The "only if" part is for free, as ∞-safety subsumes by definition T-safety. For the "if" direction, the constructed K in Eq. (12) suffices as an upper bound of $\mathrm{e}^{-\alpha t}\,\|\boldsymbol{\xi}_{\phi'}(t)\|$, and hence by Theorem 2, $\|\boldsymbol{\xi}_{\phi}(t)\| \leq \hat{K}\mathrm{e}^{\alpha t}$ for any $t \geq 0$ and $\boldsymbol{\phi}$ constrained by $\mathcal{X}$. As a consequence, it suffices to show that T^* given by Eq. (13) is finite, which then by definition implies that system (3) is safe over $t > T^*$. Note that the assumption $\mathbf{0} \notin \overline{\mathcal{U}}$ implies that there exists a ball $\mathcal{B}(\mathbf{0},\delta)$ such that $\mathcal{B}(\mathbf{0},\delta)\cap\mathcal{U} = \emptyset$. Moreover, $\hat{K}\mathrm{e}^{\alpha t}$ is strictly monotonically decreasing w.r.t. t, and thus $T = \max\{0, \ln(\delta/\hat{K})/\alpha\}$ is an upper bound[3] of T^*, which further implies $T^* < \infty$. □

Example 2 (PD-controller [17]*).* Consider a PD-controller with linear dynamics defined, for $t \geq 0$, as

$$\dot{y}(t) = v(t); \quad \dot{v}(t) = -\kappa_p\,(y(t-r) - y^*) - \kappa_d v(t-r), \quad (14)$$

which controls the position y and velocity v of an autonomous vehicle by adjusting its acceleration according to the current distance to a reference position y^*. A constant time

[2] We rule out the case of $\alpha = 0$, which renders the integral in Eq. (12) divergent.

[3] Note that the larger δ is, the tighter bound T will be.

delay r is introduced to model the time lag due to sensing, computation, transmission, and/or actuation. We instantiate the parameters following [17] as $\kappa_p = 2$, $\kappa_d = 3$, $y^* = 1$, and $r = 0.35$. The system described by Eq. (14) then has one equilibrium at $(1; 0)$, which shares equivalent stability with the zero equilibrium of the following system, with $\hat{y} = y - 1$ and $\hat{v} = v$:

$$\dot{\hat{y}}(t) = \hat{v}(t); \quad \dot{\hat{v}}(t) = -2\hat{y}(t-r) - 3\hat{v}(t-r). \tag{15}$$

Suppose we are interested in exploiting the safety property of the system (15) in an unbounded time domain, relative to the set of initial states $\mathcal{X} = [-0.1, 0.1] \times [0, 0.1]$ and the set of unsafe states $\mathcal{U} = \{(\hat{y}; \hat{v}) \mid |\hat{y}| > 0.2\}$. Following our construction process, we obtain automatically some key arguments (depicted in Fig. 1) as $\alpha = -0.5$, $M = 11.9125$, $K = 7.59162$ and $\hat{K} = 2.21103$, which consequently yield $T^* = 4.80579$ s. By Theorem 3, the unbounded safety verification problem thus is reduced to a T-bounded one for any $T > T^*$, inasmuch as ∞-safety is equivalent to T-safety for the underlying dynamics.

$[-\hat{K}\mathrm{e}^{\alpha t}, \hat{K}\mathrm{e}^{\alpha t}]^n$ in Eq. (13) can be viewed as an overapproximation of all trajectories originating from $\mathcal{X}$. As shown in the right part of Fig. 1, this overapproximation, however, is obviously too conservative to be utilized in proving or disproving almost any safety specifications of practical interest. The contribution of our approach lies in the reduction of unbounded verification problems to their bounded counterparts, thereby yielding a quantitative time bound T^* that substantially "trims off" the verification efforts pertaining to $t > T^*$. The derived T-safety verification task can be tackled effectively by methods dedicated to bounded verification of DDEs of the form (3), or more generally, (1), e.g., approaches in [17] and [4].

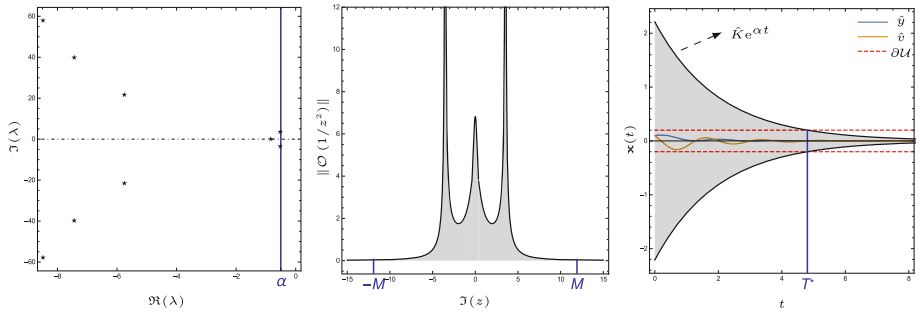


Fig. 1. Left: the identified rightmost roots of $h(z)$ in DDE-BIFTOOL and an upper bound $\alpha = -0.5$ such that $\max_{\lambda \in \sigma} \Re(\lambda) < \alpha < 0$; Center: $M = 11.9125$ that suffices to split and hence upper-bound the improper integral $\int_{-\infty}^{\infty} \left\| \mathcal{O}\left(1/z^2\right) \right\| \mathrm{d}\nu$ in Eq. (11); Right: the obtained time instant $T^* = 4.80579$ s guaranteeing the equivalence of ∞-safety and T-safety of the PD-controller, for any $T > T^*$.

4 Nonlinear Dynamics

In this section, we address a more general form of dynamics featuring substantial nonlinearity, by resorting to linearization techniques and thereby establishing a quantitative stability criterion, analogous to the linear case, for nonlinear delayed dynamics.

Consider a singly delayed version of Eq. (1):

$$\begin{cases} \dot{\mathbf{x}}(t) = \boldsymbol{f}(\mathbf{x}(t), \mathbf{x}(t-r)), & t \in [0, \infty) \\ \mathbf{x}(t) = \boldsymbol{\phi}(t), & t \in [-r, 0] \end{cases} \tag{16}$$

with $\boldsymbol{f}$ being a nonlinear vector field involving possibly non-polynomial functions. Let

$$\boldsymbol{f}(\mathbf{x}, \mathbf{y}) = A\mathbf{x} + B\mathbf{y} + \boldsymbol{g}(\mathbf{x}, \mathbf{y}), \text{ with } A = \boldsymbol{f}_{\mathbf{x}}(\mathbf{0}, \mathbf{0}), B = \boldsymbol{f}_{\mathbf{y}}(\mathbf{0}, \mathbf{0}),$$

where $\boldsymbol{f}_{\mathbf{x}}$ and $\boldsymbol{f}_{\mathbf{y}}$ are the Jacobian matrices of $\boldsymbol{f}$ in terms of $\mathbf{x}$ and $\mathbf{y}$, respectively; $\boldsymbol{g}$ is a vector-valued, high-order term whose Jacobian matrix at $(\mathbf{0}, \mathbf{0})$ is O.

By dropping the high-order term $\boldsymbol{g}$ in $\boldsymbol{f}$, we get the linearized counterpart of Eq. (16):

$$\begin{cases} \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{x}(t-r), & t \in [0, \infty) \\ \mathbf{x}(t) = \boldsymbol{\phi}(t), & t \in [-r, 0] \end{cases} \tag{17}$$

which falls in the scope of linear dynamics specified in Eq. (3), and therefore is associated with a characteristic equation of the same form as that in Eq. (4). Equation (17) will be in the sequel referred to as the linearization of Eq. (16) at the steady state $\mathbf{0}$, and σ is used to denote the spectrum of the characteristic equation corresponding to Eq. (17).

In light of the well-known Hartman-Grobman theorem [18,20] in the realm of dynamical systems, the local behavior of a nonlinear dynamical system near a (hyperbolic) equilibrium is qualitatively the same as that of its linearization near this equilibrium. The following statement uncovers the connection between the locally asymptotic behavior of a nonlinear system and the spectrum of its linearization:

Theorem 4 (Locally exponential stability [6,36]). *Suppose* $\max_{\lambda \in \sigma} \Re(\lambda) < \alpha < 0$. *Then* $\mathbf{x} = \mathbf{0}$ *is a locally exponentially stable equilibrium of the nonlinear systems* (16). *In fact, there exists* $\delta > 0$ *and* $K > 0$ *such that*

$$\|\boldsymbol{\phi}\| \leq \delta \implies \|\boldsymbol{\xi}_{\boldsymbol{\phi}}(t)\| \leq K \|\boldsymbol{\phi}\| \, \mathrm{e}^{\alpha t/2}, \quad \forall t \geq 0,$$

where $\boldsymbol{\xi}_{\boldsymbol{\phi}}(t)$ *is the solution to Eq.* (16). *If* $\Re(\lambda) > 0$ *for some* λ *in* σ, *then* $\mathbf{x} = \mathbf{0}$ *is unstable.*

Akin to the linear case, Theorem 4 establishes an existential guarantee that the solution to the nonlinear delayed dynamics approaches the zero equilibrium exponentially for initial conditions within a δ-neighborhood of this equilibrium. The need of

constructing α, K and δ quantitatively in Theorem 4, as essential to our automatic verification approach, invokes again the fundamental solution $\boldsymbol{\xi}_{\phi'}(t)$ to the linearized dynamics in Eq. (17):

Lemma 5 (Variation-of-constants [19,36]). *Consider nonhomogeneous systems of the form*

$$\begin{cases} \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{x}(t-r) + \boldsymbol{\eta}(t), & t \in [0,\infty) \\ \mathbf{x}(t) = \boldsymbol{\phi}(t), & t \in [-r,0] \end{cases} \tag{18}$$

Let $\boldsymbol{\xi}_{\phi}(t)$ be the solution to Eq. (18). Denote by $\boldsymbol{\xi}_{\phi'}(t)$ the solution that satisfies Eq. (17) for $t \geq 0$ and satisfies a variation of the initial condition as $\boldsymbol{\phi}'(0) = I$ and $\boldsymbol{\phi}'(t) = O$ for all $t \in [-r, 0)$. Then for $t \geq 0$,

$$\boldsymbol{\xi}_{\phi}(t) = \boldsymbol{\xi}_{\phi'}(t)\boldsymbol{\phi}(0) + \int_0^t \boldsymbol{\xi}_{\phi'}(t-\tau)B\boldsymbol{\phi}(\tau - r)\,\mathrm{d}\tau + \int_0^t \boldsymbol{\xi}_{\phi'}(t-\tau)\boldsymbol{\eta}(\tau)\,\mathrm{d}\tau, \tag{19}$$

where $\boldsymbol{\phi}$ is extended to $[-r,\infty)$ with $\boldsymbol{\phi}(t) = 0$ for $t > 0$.

In what follows, we give a constructive quantitative estimation of the solutions to nonlinear dynamics, which admits a reduction of the problem of constructing an exponential upper bound of a nonlinear system to that of its linearization, as being immediately evident from the constructive proof.

Theorem 5 (Exponential estimation). *Suppose that $\max_{\lambda\in\sigma} \Re(\lambda) < \alpha < 0$. Then there exist $K > 0$ and $\delta > 0$ such that $\|\boldsymbol{\xi}_{\phi'}(t)\| \leq K\mathrm{e}^{\alpha t}$ for any $t \geq 0$, and*

$$\|\boldsymbol{\phi}\| \leq \delta \implies \|\boldsymbol{\xi}_{\phi}(t)\| \leq K\mathrm{e}^{-r\alpha}\left(1 + \|B\|\int_0^r \mathrm{e}^{-\alpha\tau}\,\mathrm{d}\tau\right)\|\boldsymbol{\phi}\|\,\mathrm{e}^{\alpha t/2}, \quad \forall t \geq 0,$$

where $\boldsymbol{\xi}_{\phi}(t)$ is the solution to nonlinear systems (16) and $\boldsymbol{\xi}_{\phi'}(t)$ is the fundamental solution to the linearized counterpart (17).

Proof. The existence of K follows directly from Eq. (7) in Theorem 2. By the variation-of-constants formula (19), we have, for $t \geq 0$,

$$\boldsymbol{\xi}_{\phi}(t) = \boldsymbol{\xi}_{\phi'}(t)\boldsymbol{\phi}(0) + \int_0^t \boldsymbol{\xi}_{\phi'}(t-\tau)B\boldsymbol{\phi}(\tau-r)\,\mathrm{d}\tau + \int_0^t \boldsymbol{\xi}_{\phi'}(t-\tau)\boldsymbol{g}(\mathbf{x}(\tau),\mathbf{x}(\tau-r))\,\mathrm{d}\tau, \tag{20}$$

where $\boldsymbol{\phi}$ is extended to $[-r,\infty)$ with $\boldsymbol{\phi}(t) = 0$ for $t > 0$. Define $\mathbf{x}_t^{\phi}(\cdot) \in \mathcal{C}_r$ as $\mathbf{x}_t^{\phi}(\theta) = \boldsymbol{\xi}_{\phi}(t+\theta)$ for $\theta \in [-r, 0]$. Then $\boldsymbol{g}(\cdot,\cdot)$ being a higher-order term yields that for any $\epsilon > 0$, there exists $\delta_\epsilon > 0$ such that $\|\mathbf{x}_t^{\phi}\| \leq \delta_\epsilon$ implies $\boldsymbol{g}(\mathbf{x}(t),\mathbf{x}(t-r)) \leq \epsilon\|\mathbf{x}_t^{\phi}\|$. Due to the fact that $\|\boldsymbol{\xi}_{\phi'}(t)\| \leq K\mathrm{e}^{\alpha t}$ and the monotonicity of $\|\boldsymbol{\xi}_{\phi'}(t)\|$ with $\alpha < 0$, we have $\|\mathbf{x}_t^{\phi'}\| \leq K\mathrm{e}^{\alpha(t-r)}$. This, together with Eq. (20), leads to

$$\begin{aligned} \|\mathbf{x}_t^{\phi}\| &\leq K\|\boldsymbol{\phi}\|\,\mathrm{e}^{\alpha(t-r)} + \int_0^r K\|B\|\,\|\boldsymbol{\phi}\|\,\mathrm{e}^{\alpha(t-r)}\mathrm{e}^{-\alpha\tau}\,\mathrm{d}\tau + \int_0^t K\mathrm{e}^{\alpha(t-r)}\mathrm{e}^{-\alpha\tau}\epsilon\|\mathbf{x}_\tau^{\phi}\|\,\mathrm{d}\tau \\ &= K\left(1 + \|B\|\int_0^r \mathrm{e}^{-\alpha\tau}\,\mathrm{d}\tau\right)\|\boldsymbol{\phi}\|\,\mathrm{e}^{\alpha(t-r)} + \epsilon K\mathrm{e}^{\alpha(t-r)}\int_0^t \mathrm{e}^{-\alpha\tau}\|\mathbf{x}_\tau^{\phi}\|\,\mathrm{d}\tau. \end{aligned}$$

Hence,

$$\mathrm{e}^{-\alpha t}\left\|\mathbf{x}_t^{\phi}\right\| \leq K\mathrm{e}^{-r\alpha}\left(1+\|B\|\int_0^r \mathrm{e}^{-\alpha\tau}\,\mathrm{d}\tau\right)\|\boldsymbol{\phi}\| + \epsilon K\mathrm{e}^{-r\alpha}\int_0^t \mathrm{e}^{-\alpha\tau}\left\|\mathbf{x}_\tau^{\phi}\right\|\mathrm{d}\tau.$$

By the Grönwall-Bellman inequality [1] we obtain

$$\mathrm{e}^{-\alpha t}\left\|\mathbf{x}_t^{\phi}\right\| \leq K\mathrm{e}^{-r\alpha}\left(1+\|B\|\int_0^r \mathrm{e}^{-\alpha\tau}\,\mathrm{d}\tau\right)\|\boldsymbol{\phi}\|\,\mathrm{e}^{\epsilon K\mathrm{e}^{-r\alpha}t}$$

and thus

$$\left\|\mathbf{x}_t^{\phi}\right\| \leq K\mathrm{e}^{-r\alpha}\left(1+\|B\|\int_0^r \mathrm{e}^{-\alpha\tau}\,\mathrm{d}\tau\right)\|\boldsymbol{\phi}\|\,\mathrm{e}^{\epsilon K\mathrm{e}^{-r\alpha}t+\alpha t}.$$

Set $\epsilon \leq -\alpha/(2K\mathrm{e}^{-r\alpha})$ and $\delta = \min\left\{\delta_\epsilon, \delta_\epsilon/\left(K\mathrm{e}^{-r\alpha}\left(1+\|B\|\int_0^r \mathrm{e}^{-\alpha\tau}\,\mathrm{d}\tau\right)\right)\right\}$. This yields, for any $t \geq 0$,

$$\|\boldsymbol{\phi}\| \leq \delta \implies \|\boldsymbol{\xi}_{\boldsymbol{\phi}}(t)\| \leq K\mathrm{e}^{-r\alpha}\left(1+\|B\|\int_0^r \mathrm{e}^{-\alpha\tau}\,\mathrm{d}\tau\right)\|\boldsymbol{\phi}\|\,\mathrm{e}^{\alpha t/2},$$

completing the proof. □

The above constructive quantitative estimation of the solutions to nonlinear dynamics gives rise to the reduction, analogous to the linear case, of unbounded verification problems to bounded ones, in the presence of a local stability criterion.

Theorem 6 (Equivalence of safety properties). *Given initial state set $\mathcal{X} \subseteq \mathbb{R}^n$ and bad states $\mathcal{U} \subseteq \mathbb{R}^n$ satisfying $\mathbf{0} \notin \overline{\mathcal{U}}$. Let σ denote the spectrum of the characteristic equation corresponding to Eq. (17). Suppose that $\max_{\lambda\in\sigma}\Re(\lambda) < \alpha < 0$, and the fundamental solution to Eq. (17) satisfies $\|\boldsymbol{\xi}_{\boldsymbol{\phi}'}(t)\| \leq K\mathrm{e}^{\alpha t}$ for any $t \geq 0$. Let $\tilde{K} = K\mathrm{e}^{-r\alpha}\left(1+\|B\|\int_0^r \mathrm{e}^{-\alpha\tau}\,\mathrm{d}\tau\right)\|\mathcal{X}\|$. Then there exists $\delta > 0$ and $T^* < \infty$, defined as*

$$T^* \mathrel{\widehat{=}} \max\{0, \inf\{T \mid \forall t > T\colon [-\tilde{K}\mathrm{e}^{\alpha t/2}, \tilde{K}\mathrm{e}^{\alpha t/2}]^n \cap \mathcal{U} = \emptyset\}\},$$

such that if $\|\mathcal{X}\| \leq \delta$, then for any $T > T^*$, the system (16) is ∞-safe iff it is T-safe.

Proof. The proof is analogous to that of Theorem 3, particularly following from the local stability property stated in Theorem 5. □

Note that for nonlinear dynamics, the equivalence of safety claimed by Theorem 6 holds on the condition that $\|\mathcal{X}\| \leq \delta$, due to the locality stemming from linearization. In fact, such a set $\mathfrak{B} \subseteq \mathbb{R}^n$ satisfying $\|\mathfrak{B}\| \leq \delta$ describes (a subset of) the basin of attraction around the local *attractor* $\mathbf{0}$, in a sense that any initial condition in $\mathfrak{B}$ will lead the trajectory eventually into the attractor. Consequently, for verification problems where $\mathcal{X} \supseteq \mathfrak{B}$, if the reachable set originating from $\mathcal{X}$ is guaranteed to be subsumed within $\mathfrak{B}$ in the time interval $[T'-r, T']$, then $T' + T^*$ suffices as a bound to avoid unbounded verification, namely for any $T > T' + T^*$, the system is ∞-safe iff it is T-safe. This is furthermore demonstrated by the following example.

Example 3 (Population dynamics [4,25]*).* Consider a slightly modified version of the delayed logistic equation introduced by G. Hutchinson in 1948 (cf. [22])

$$\dot{N}(t) = N(t)[1 - N(t-r)], \quad t \geq 0, \tag{21}$$

which is used to model a single population whose percapita rate of growth $\dot{N}(t)/N(t)$ depends on the population size r time units in the past. This would be a reasonable model for a population that features a significant minimum reproductive age or depends on a resource, like food, needing time to grow and thus to recover its availability.

If we change variables, putting $u = N - 1$, then Eq. (21) becomes the famous Wright's equation (see [44]):

$$\dot{u}(t) = -u(t-r)[1 + u(t)], \quad t \geq 0. \tag{22}$$

The steady state $N = 1$ is now $u = 0$. We instantiate the verification problem of Eq. (22) over $[-r, \infty)$ as $\mathcal{X} = [-0.2, 0.2]$, $\mathcal{U} = \{u \mid |u| > 0.6\}$, under a constant delay $r = 1$. Note that delay-independent Lyapunov techniques, e.g. [32], cannot solve this problem, since Wright's conjecture [44], which has been recently proven in [40], together with corollaries thereof implies that there does not exist a Lyapunov functional guaranteeing absolute stability of Eq. (22) with arbitrary constant delays. To achieve an exponential estimation, we first linearize the dynamics by dropping the nonlinearity $u(t)u(t-r)$ thereof:

$$\dot{v}(t) = -v(t-1), \quad t \geq 0. \tag{23}$$

Following our constructive approach, we obtain automatically for Eq. (23) $\alpha = -0.3$ (see the left of Fig. 2), $M = 2.69972$, $K = 3.28727$, and thereby for Eq. (22) $\delta = 0.00351678$, $\tilde{K} = 0.0338039$ and $T^* = 0$ s. It is worth highlighting that by the bounded verification method in [17], with Taylor models of the order 5, an overapproximation Ω of the reachable set w.r.t. system (22) over the time interval $[14.5, 15.5]$ was verified to be enclosed in the δ-neighborhood of $\mathbf{0}$, i.e., $\|\Omega\| \leq \delta$, yet escaped from this region around $t = 55.3$ s, and tended to diverge soon, as depicted in the right part of Fig. 2, and thus cannot prove unbounded safety properties. However, with our result of $T^* = 0$s and the fact that Ω over $[-1, 15.5]$ is disjoint with $\mathcal{U}$, we are able to claim safety of the underlying system over an infinite time domain.

DDEs with Multiple Different Delays. Delay differential equations with multiple fixed discrete delays are extensively used in the literature to model practical systems where components coupled with different time lags coexist and interact with each other. We remark that previous theorems on exponential estimation and equivalence of safety w.r.t. cases of single delay extend immediately to systems of the form (1) with almost no change, except for replacing $\|B\| \, \mathrm{e}^{-r\alpha}$ with $\sum_{i=1}^{k} \|A_i\| \, \mathrm{e}^{-r_i\alpha}$ and $\|B\|$ with $\sum_{i=1}^{k} \|A_i\|$, where A_i denotes the matrix attached to $\mathbf{x}(t - r_i)$ in the linearization. For a slightly modified form of the variation-of-constants formula under multiple delays, we refer the readers to Theorem 1.2 in [19].

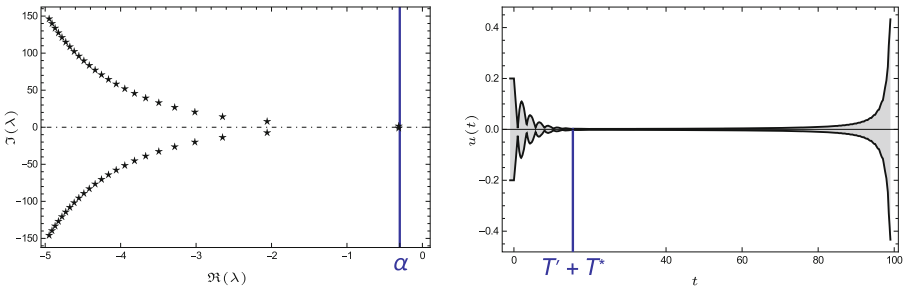


Fig. 2. Left: the identified rightmost eigenvalues of $h(z)$ and an upper bound $\alpha = -0.5$ such that $\max_{\lambda \in \sigma} \Re(\lambda) < \alpha < 0$; Right: overapproximation of the reachable set of the system (22) produced by the method in [17] using Taylor models for bounded verification. Together with this overapproximation we prove the equivalence of ∞-safety and T-safety of the system, for any $T > (T' + T^*) = 15.5\,\text{s}$.

5 Implementation and Experimental Results

To further investigate the scalability and efficiency of our constructive approach, we have carried out a prototypical implementation[4] in Wolfram MATHEMATICA, which was selected due to its built-in primitives for integration and matrix operations. By interfacing with DDE-BIFTOOL[5] (in MATLAB or GNU OCTAVE) for identifying the rightmost characteristic roots of linear (or linearized) DDEs, our implementation computes an appropriate T^* that admits a reduction of unbounded verification problems to bounded ones. A set of benchmark examples from the literature has been evaluated on a 3.6 GHz Intel Core-i7 processor with 8 GB RAM running 64-bit Ubuntu 16.04. All computations of T^* were safely rounded and finished within 6 s for any of the examples, including Examples 2 and 3. In what follows, we demonstrate in particular the applicability of our technique to DDEs featuring non-polynomial dynamics, high dimensionality and multiple delays.

Example 4 (Disease pathology [25,27,32]*).* Consider the following non-polynomial DDE for $t \geq 0$:

$$\dot{p}(t) = \frac{\beta \theta^n p(t-r)}{\theta^n + p^n(t-r)} - \gamma p(t), \tag{24}$$

where $p(t)$ is positive and indicates the number of mature blood cells in circulation, while r models the delay between cell production and cell maturation. We consider the case $\theta = 1$ as in [32]. Constants are instantiated as $n = 1$, $\beta = 0.5$, $\gamma = 0.6$ and $r = 0.5$. The unbounded verification problem of Eq. (24) over $[-r, \infty)$ is configured as $\mathcal{X} = [0, 0.2]$ and $\mathcal{U} = \{p \mid |p| > 0.3\}$. Then the linearization of Eq. (24) reads

$$\dot{p}(t) = -0.6p(t) + 0.5p(t-0.5). \tag{25}$$

[4] http://lcs.ios.ac.cn/~chenms/tools/UDDER.tar.bz2.
[5] http://ddebiftool.sourceforge.net/.

With $\alpha = -0.07$ obtained from DDE-BIFTOOL, our implementation produces for Eq. (25) the values $M = 2.23562$, $K = 1.75081$, and thereby for Eq. (24) $\delta = 0.0163426$, $\tilde{K} = 0.0371712$ and $T^* = 0\,\mathrm{s}$. Thereafter by the bounded verification method in [17], with Taylor models of the order 5, an overapproximation of the reachable set w.r.t. system (24) over the time interval $[25.45, 25.95]$ was verified to be enclosed in the δ-neighborhood of $\mathbf{0}$. This fact, together with $T^* = 0\,\mathrm{s}$ and the overapproximation on $[-0.5, 25.95]$ being disjoint with $\mathcal{U}$, yields safety of the system (24) over $[-0.5, \infty)$.

Example 5 (Gene regulation [12,36]*).* To examine the scalability of our technique to higher dimensions, we recall an instantiation of Eq. (2) by setting $n = 5$, namely with 5 state components $\mathbf{x} = (x_1; \ldots; x_5)$ and 5 delay terms $\mathbf{r} = (0.1; 0.2; 0.4; 0.8; 1.6)$ involved, $g(x) = -x$, $\beta_j = 1$ for $j = 1, \ldots, 5$, $\mathcal{X} = \mathcal{B}((1;1;1;1;1), 0.2)$ and $\mathcal{U} = \{\mathbf{x} \mid |x_1| > 1.5\}$. With $\alpha = -0.04$ derived from DDE-BIFTOOL, our implementation returns $M = 64.264$, $K = 4.42207$, $\hat{K} = 49.1463$ and $T^* = 87.2334\,\mathrm{s}$, thereby yielding the equivalence of ∞-safety to T-safety for any $T > T^*$. Furthermore, the safety guarantee issued by the bounded verification method in [4] based on rigorous simulations under $T = 88\,\mathrm{s}$ suffices to prove safety of the system over an infinite time horizon.

6 Conclusion

We have presented a constructive method, based on linearization and spectral analysis, for computing a delay-dependent, exponentially decreasing upper bound, if existent, that encloses trajectories of a DDE originating from a certain set of initial functions. We showed that such an enclosure facilitates a reduction of the verification problem over an unbounded temporal horizon to a bounded one. Preliminary experimental results on a set of representative benchmarks from the literature demonstrate that our technique effectively extends the scope of existing bounded verification techniques to unbounded verification tasks.

Peeking into future directions, we plan to exploit a tight integration of our technique into several automatic tools dedicated to bounded verification of DDEs, as well as more permissive forms of stabilities, e.g. asymptotical stability, that may admit a similar reduction-based idea. An extension of our method to deal with more general forms of DDEs, e.g., with time-varying, or distributed (i.e., a weighted average of) delays, will also be of interest. Additionally, we expect to refine our enclosure of system trajectories by resorting to a topologically finite partition of the initial set of functions.

References

1. Bellman, R.: The stability of solutions of linear differential equations. Duke Math. J. **10**(4), 643–647 (1943)
2. Bellman, R.E., Cooke, K.L.: Differential-difference equations. Technical Report R-374-PR, RAND Corporation, Santa Monica, California, January 1963
3. Breda, D., Maset, S., Vermiglio, R.: Computing the characteristic roots for delay differential equations. IMA J. Numer. Anal. **24**(1), 1–19 (2004)

4. Chen, M., Fränzle, M., Li, Y., Mosaad, P.N., Zhan, N.: Validated simulation-based verification of delayed differential dynamics. In: Fitzgerald, J., Heitmeyer, C., Gnesi, S., Philippou, A. (eds.) FM 2016. LNCS, vol. 9995, pp. 137–154. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48989-6_9
5. Cooke, K.L.: Stability analysis for a vector disease model. Rocky Mt. J. Math. **9**(1), 31–42 (1979)
6. Diekmann, O., van Gils, S., Lunel, S., Walther, H.: Delay Equations: Functional-, Complex-, and Nonlinear Analysis. Applied Mathematical Sciences. Springer, New York (2012). https://doi.org/10.1007/978-1-4612-4206-2
7. Donzé, A., Maler, O.: Systematic simulation using sensitivity analysis. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 174–189. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71493-4_16
8. Driver, R.: Ordinary and Delay Differential Equations. Applied Mathematical Sciences. Springer, New York (1977). https://doi.org/10.1007/978-1-4684-9467-9
9. Duggirala, P.S., Mitra, S., Viswanathan, M.: Verification of annotated models from executions. In: EMSOFT 2013, pp. 26:1–26:10 (2013)
10. Engelborghs, K., Luzyanina, T., Roose, D.: Numerical bifurcation analysis of delay differential equations using DDE-BIFTOOL. ACM Trans. Math. Softw. **28**(1), 1–21 (2002)
11. Engelborghs, K., Roose, D.: On stability of LMS methods and characteristic roots of delay differential equations. SIAM J. Numer. Anal. **40**(2), 629–650 (2002)
12. Fall, C.P., Marland, E.S., Wagner, J.M., Tyson, J.J. (eds.): Computational Cell Biology, vol. 20. Springer, New York (2002)
13. Fort, J., Méndez, V.: Time-delayed theory of the neolithic transition in Europe. Phys. Rev. Lett. **82**(4), 867 (1999)
14. Fränzle, M., Chen, M., Kröger, P.: In memory of Oded Maler: automatic reachability analysis of hybrid-state automata. ACM SIGLOG News **6**(1), 19–39 (2019)
15. Gan, T., Chen, M., Li, Y., Xia, B., Zhan, N.: Reachability analysis for solvable dynamical systems. IEEE Trans. Automat. Contr. **63**(7), 2003–2018 (2018)
16. Girard, A., Pappas, G.J.: Approximate bisimulation: a bridge between computer science and control theory. Eur. J. Control **17**(5–6), 568–578 (2011)
17. Goubault, E., Putot, S., Sahlmann, L.: Inner and outer approximating flowpipes for delay differential equations. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10982, pp. 523–541. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96142-2_31
18. Grobman, D.M.: Homeomorphism of systems of differential equations. Doklady Akademii Nauk SSSR **128**(5), 880–881 (1959)
19. Hale, J., Lunel, S.: Introduction to Functional Differential Equations. Applied Mathematical Sciences. Springer, New York (1993). https://doi.org/10.1007/978-1-4612-4342-7
20. Hartman, P.: A lemma in the theory of structural stability of differential equations. Proc. Am. Math. Soc. **11**(4), 610–620 (1960)
21. Huang, Z., Fan, C., Mitra, S.: Bounded invariant verification for time-delayed nonlinear networked dynamical systems. Nonlinear Anal. Hybrid Syst. **23**, 211–229 (2017)
22. Hutchinson, G.E.: Circular causal systems in ecology. Ann. N. Y. Acad. Sci. **50**(4), 221–246 (1948)
23. Ikeda, K., Matsumoto, K.: High-dimensional chaotic behavior in systems with time-delayed feedback. Phys. D Nonlinear Phenom. **29**(1–2), 223–235 (1987)
24. Krasovskiĭ, N.: Stability of Motion: Applications of Lyapunov's Second Method to Differential Systems and Equations with Delay. Studies in Mathematical Analysis and Related Topics. Stanford University Press, Stanford (1963)
25. Kuang, Y.: Delay Differential Equations: With Applications in Population Dynamics. Mathematics in Science and Engineering. Elsevier Science, Amsterdam (1993)

26. Levine, W.S.: The Control Handbook: Control System Fundamentals. Electrical Engineering Handbook, 2nd edn. CRC Press, Boca Raton (2010)
27. Mackey, M.C., Glass, L.: Oscillation and chaos in physiological control systems. Science **197**(4300), 287–289 (1977)
28. Mallet-Paret, J., Sell, G.R.: The Poincaré-Bendixson theorem for monotone cyclic feedback systems with delay. J. Differ. Equ. **125**, 441–489 (1996)
29. Nazier Mosaad, P., Fränzle, M., Xue, B.: Temporal logic verification for delay differential equations. In: Sampaio, A., Wang, F. (eds.) ICTAC 2016. LNCS, vol. 9965, pp. 405–421. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46750-4_23
30. Myshkis, A.D.: Lineare Differentialgleichungen mit nacheilendem Argument, vol. 17. VEB Deutscher Verlag der Wissenschaften (1955)
31. Nahhal, T., Dang, T.: Test coverage for continuous and hybrid systems. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 449–462. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_47
32. Peet, M., Lall, S.: Constructing Lyapunov functions for nonlinear delay-differential equations using semidefinite programming. In: Proceedings of NOLCOS, pp. 381–385 (2004)
33. Pola, G., Pepe, P., Benedetto, M.D.D.: Symbolic models for time-varying time-delay systems via alternating approximate bisimulation. Int. J. Robust Nonlinear Control **25**, 2328–2347 (2015)
34. Pola, G., Pepe, P., Benedetto, M.D.D., Tabuada, P.: Symbolic models for nonlinear time-delay systems using approximate bisimulations. Syst. Control Lett. **59**(6), 365–373 (2010)
35. Prajna, S., Jadbabaie, A.: Methods for safety verification of time-delay systems. In: CDC 2005, pp. 4348–4353 (2005)
36. Smith, H.: An Introduction to Delay Differential Equations with Applications to the Life Sciences, vol. 57. Springer, New York (2011). https://doi.org/10.1007/978-1-4419-7646-8
37. Strzeboński, A.: Cylindrical decomposition for systems transcendental in the first variable. J. Symb. Comput. **46**(11), 1284–1290 (2011)
38. Szydłowski, M., Krawiec, A., Toboła, J.: Nonlinear oscillations in business cycle model with time lags. Chaos Solitons Fractals **12**(3), 505–517 (2001)
39. Vajta, M.: Some remarks on padé-approximations. In: Proceedings of the 3rd TEMPUS-INTCOM Symposium, vol. 242 (2000)
40. van den Berg, J.B., Jaquette, J.: A proof of Wright's conjecture. J. Differ. Equ. **264**(12), 7412–7462 (2018)
41. Volterra, V.: Une théorie mathématique de la lutte pour la vie (1927)
42. Volterra, V.: Sur la théorie mathématique des phénomenes héréditaires. Journal de mathématiques pures et appliquées **7**, 249–298 (1928)
43. Vyhlídal, T.: Analysis and synthesis of time delay system spectrum. Ph.D. dissertation, Czech Technical University in Prague (2003)
44. Wright, E.M.: A non-linear difference-differential equation. J. Reine Angew. Math. **66–87**, 1955 (1955)
45. Wulf, V., Ford, N.J.: Numerical hopf bifurcation for a class of delay differential equations. J. Comput. Appl. Math. **115**(1–2), 601–616 (2000)
46. Xue, B., Mosaad, P.N., Fränzle, M., Chen, M., Li, Y., Zhan, N.: Safe over- and under-approximation of reachable sets for delay differential equations. In: Abate, A., Geeraerts, G. (eds.) FORMATS 2017. LNCS, vol. 10419, pp. 281–299. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65765-3_16
47. Zou, L., Fränzle, M., Zhan, N., Mosaad, P.N.: Automatic verification of stability and safety for delay differential equations. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9207, pp. 338–355. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21668-3_20

Author Index