EPTCS 145

Proceedings of the

1st International Workshop on

Synthesis of Continuous Parameters

Grenoble, France, 6th April 2014

Edited by: Étienne André and Goran Frehse

Published: 31st March 2014
DOI: 10.4204/EPTCS.145
ISSN: 2075-2180
Open Publishing Association

Preface

Étienne André
Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France
Etienne.Andre@univ-paris13.fr

Goran Frehse
Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France
Goran.Frehse@imag.fr

This volume contains the proceedings of the 1st International Workshop on Synthesis of Continuous Parameters (SynCoP'14). The workshop was held in Grenoble, France on April 6th, 2014, as a satellite event of the 17th European Joint Conferences on Theory and Practice of Software (ETAPS'14).

SynCoP aims at bringing together researchers working on parameter synthesis for systems with continuous variables, where the parameters consist of a (usually dense) set of constant values. Synthesis problems for such parameters arise for real-time, hybrid or probabilistic systems in a large variety application domains. A parameter could be, e.g., a delay in a real-time system, or a reaction rate in a biological cell model. The objective of the synthesis problem is to identify suitable parameters to achieve desired behavior, or to verify the behavior for a given range of parameter values.

The scientific subject of the workshop covers (but is not limited to) the following areas:

- parameter synthesis,
- parametric model checking,
- robustness analysis,
- parametric logics, decidability and complexity issues,
- formalisms such as parametric timed and hybrid automata, parametric time(d) Petri nets, parametric probabilistic automata, parametric Markov decision processes, and
- applications to major areas of computer science and control engineering.

Program

This volume contains seven contributions: two invited talks and five regular papers. The two invited talks are:

- Integer Parameter Synthesis for Timed Automata (Didier Lime)
- Parameter Synthesis for Signal Temporal Logic (Alexandre Donzé)

The workshop received seven submissions, five of which were accepted. Each regular paper was reviewed by at least three different reviewers. The five regular papers are:

- MTL-Model Checking of One-Clock Parameterized Timed Automata is Undecidable (Karin Quaas)
- Probabilistic Bisimulations for PCTL Model Checking of Interval MDPs (Vahid Hashemi, Hassan Hatefi and Jan Krčál)

É. André and G. Frehse (Eds.): 1st International Workshop
on Synthesis of Continuous Parameters (SynCoP'14).
EPTCS 145, 2014, pp. i–iii, doi:10.4204/EPTCS.145.0

- Setting Parameters for Biological Models with ANIMO (Stefano Schivo, Jetse Scholma, Marcel Karperien, Janine N. Post, Jaco Van De Pol and Rom Langerak)
- Toward Parametric Timed Interfaces for Real-Time Components (Giuseppe Lipari, Youcheng Sun, Étienne André and Laurent Fribourg)
- Worst-case Throughput Analysis for Parametric Rate and Parametric Actor Execution Time Scenario-Aware Dataflow Graphs (Mladen Skelin, Marc Geilen, Francky Catthoor and Sverre Hendseth)

Furthermore, one informal presentation was made at the workshop:

- Parameter Synthesis using Paralleltopic Enclosure (Thao Dang, Tommaso Dreossi, Carla Piazza)

Support and Acknowledgement

SynCoP 2014 is partially supported by Verimag, LIPN, Université Paris 13, and CNRS GDR IM. We would like to thank these various entities for their generous financial and organizational support, thanks to which the workshop was able to sponsor two invited speakers and two students attending the workshop.

SynCoP has been organized as a satellite event of ETAPS'14. We thank the authors for their contributions, the program committee members for reviewing and selecting the papers, and the ETAPS organizing committee Axel Legay, Ylies Falcone, Saddek Bensalem and Marius Bozga for their support. We would also like to thank Laurent Fribourg, Laure Petrucci, and Gilles Villard.

Finally, we would like to thank the editorial board of the Electronic Proceedings in Theoretical Computer Science, and in particular Editor-in-Chief Rob van Glabbeek for his support.

In Villetaneuse and Grenoble,

Étienne André and Goran Frehse

Program Committee

Organizers and Program Chairs

Étienne André	Villetaneuse, France
Goran Frehse	Grenoble, France

Program Committee

Eugene Asarin	Paris, France
Alessandro Cimatti	Trento, Italy
Alexandre Donzé	Berkeley, USA
Georgios Fainekos	Arizona, USA
Laurent Fribourg	Cachan, France
Antoine Girard	Grenoble, France
Kim Larsen	Ålborg, Denmark
Yang Liu	Singapore
Olivier H. Roux	Nantes, France
Sriram Sankaranarayanan	Boulder, USA
Ashish Tiwari	USA
Farn Wang	Taipei, Taiwan

Table of Contents

Integer Parameter Synthesis for Timed Automata

Aleksandra Jovanović
Department of Computer Science
University of Oxford, United Kingdom
Aleksandra.Jovanovic@cs.ox.ac.uk

Didier Lime Olivier H. Roux
École Centrale de Nantes, IRCCyN UMR CNRS 6597
Nantes, France
Didier.Lime@ec-nantes.fr Olivier-h.Roux@irccyn.ec-nantes.fr

In this talk, we address the problem of parametric verification for real-time systems. The correct design of such systems is certainly an important and challenging issue. Introducing parameters in the verification process has a manifold benefit: it makes it more useful by providing the designer with more information than the mere satisfaction of properties, it also makes it more flexible by allowing verification at an earlier stage of the design process when not so many features are fixed. It may finally make the verification process more robust to small changes in the specification.

The main problem encountered in this setting is that parametric verification for timed systems is very difficult. Starting with the seminal results of Alur, Henzinger and Vardi [1], this field mostly features a long list of undecidable problems, to which we will add some more in this talk.

Subclassing the general model of parametric timed automata (PTA) permits to retrieve some decidability, like for PTA with one parametric clock [1] or L/U automata [4, 2]. The latter are tailored to the obtain the decidability of the existence of parameter values that make reachability decidable, but will show that the actual synthesis of those parameters is still mostly out of reach.

To overcome these difficulties, we propose a different approach, in which we consider that the parameters should take bounded integer values. We argue that this is not such a restrictive setting, since the bound can be arbitrarily large and since real-life features like transmission times, watchdog durations, activation periods of tasks are reasonably specified by integers, or at least rationals that can be made integers with adequate scaling.

In this setting most problems are obviously decidable, including actual synthesis, since one need only enumerate all possible parameter values and perform non-parametric verification for all of them. This is however not what we want to do in practice, since it is certain to be very inefficient for large (absolute values of the) bounds on parameters. Instead we propose, a symbolic polyhedra-based computation, in the spirit of the classical symbolic semi-algorithms on hybrid systems [3], and making use of the notion of integer hull. We will also discuss the extent of the underapproximation obtained through these symbolic algorithms. These symbolic algorithms have been implemented in our tool ROMEO [5] and we will provide some benchmarking of the proposed techniques.

Finally, we will show that the actual worst-case complexity of problems in this bounded integer parametric setting is (theoretically) very close to that of the corresponding non-parametric versions, and that lifting either the integer or bounded assumption on parameter values leads to undecidability.

É. André and G. Frehse (Eds.): 1st International Workshop
on Synthesis of Continuous Parameters (SynCoP'14).
EPTCS 145, 2014, pp. 1–2, doi:10.4204/EPTCS.145.1

References

[1] Rajeev Alur, Thomas A. Henzinger & Moshe Y. Vardi (1993): *Parametric Real-time Reasoning*. In: *ACM Symposium on Theory of Computing*, pp. 592–601. Available at http://dx.doi.org/10.1145/167088.167242.

[2] Laura Bozzelli & Salvatore La Torre (2009): *Decision problems for lower/upper bound parametric timed automata*. *Formal Methods in System Design* 35(2), pp. 121–151. Available at http://dx.doi.org/10.1007/s10703-009-0074-0.

[3] Thomas A. Henzinger, Pei-Hsin Ho & Howard Wong-Toi (1997): *HyTech: A model checker for hybrid systems*. In Orna Grumberg, editor: *Computer Aided Verification*, *Lecture Notes in Computer Science* 1254, Springer Berlin Heidelberg, pp. 460–463. Available at http://dx.doi.org/10.1007/3-540-63166-6_48.

[4] Thomas Hune, Judi Romijn, Marielle Stoelinga & Frits Vaandrager (2002): *Linear Parametric Model Checking of Timed Automata*. *Journal of Logic and Algebraic Programming* 52-53, pp. 183–220. Available at http://dx.doi.org/10.1016/S1567-8326(02)00037-1.

[5] Didier Lime, Olivier H. Roux, Charlotte Seidner & Louis-Marie Traonouez (2009): *Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches*. In: *15th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, *LNCS* 5505, Springer, York, UK, pp. 54–57. Available at http://dx.doi.org/10.1007/978-3-642-00768-2_6.

Parameter Synthesis for Signal Temporal Logic

Alexandre Donzé
University of California, Berkeley
EECS Department, Cory Hall
94720 Berkeley, CA, USA
donze@berkeley.edu

In the domain of model-based design using formal methods and tools, synthesis appears as the Holy Grail. Given a set of specifications, it supposedly allows to generate automatically the design of a system which is correct-by-construction with respect to those specifications. In order to become a reality, synthesis requires to be casted in strict and precise framework. Early investigation have focused on finite state machines and linear temporal logics (LTL) but remained for a long time a purely theoretical exploration due to a seemingly unavoidable astronomical complexity of the synthesis algorithm. Nevertheless, a polynomial-time algorithm was eventually developped for a subset of LTL [12] which opened the way for practical implementations and applications in various fields: digital circuits[2], robotics [6], supervisory control [11], etc.

Synthesis for more complex systems, in particular hybrid systems involving continuous dynamics together with discrete components, brings yet additional challenges. For these systems, the use of a specification language adapted to properties of dense-time real-valued signal , namely Signal Temporal Logic (STL) [9], has been recently advocated. However, synthesizing a hybrid system from STL specifications is by large still an open problem. The difficulty of this problem can be mitigated by the introduction of parameterizations, both of the system to be synthesized and for the input specification. An appropriate parameteriterization of the system allows to reduce synthesis to the problem of finding a valuation for finite set of parameters for which the system's behaviors satisfy the specifications. In the case of STL, recently introduced *quantitative semantics* [7, 5] can be leveraged to solve this problem. Indeed, when dealing with continuous dynamics and numerical quantities, yes/no answers provide only partial information and can be augmented with quantitative information about the satisfaction to provide a better basis for decision making. The principle is to map parameters to some (signed) distance from satisfying the specification. Parameter synthesis can then be reduced to an optimization problem with such distance as a cost function.

Introducing parameters in STL specifications leads to PSTL fomulas [1]. Parameter synthesis for PSTL is dual to the problem of STL synthesis for a parameterized system. It is especially relevant in case the initial specification reveals to be infeasible, or more generally if the specification has to be revised after a first synthesis attempt, which is very frequent in practice in synthesis. It is usually the case indeed that the design process consists of actual co-design of a system and its specifications rather than a waterfall process from the specification to the design, even considering that an ideal synthesis tool is available. In [1] and later in [8], the problem of synthesizing parameter valuations for a PSTL formula and a (set of) execution traces has been considered. It was shown to be undecidable, however a general optimization problem formulation leveraging again quantitative semantics can be applied. Moreover, situations where efficient algorithms apply have been characterized. More specifically, when the parameters to be instantiated are *monotic* with respect the satisfaction of the formula, then generalized binary search can be used to find *tight* valuations, i.e., satisfying valuations which are close to the boundary

É. André and G. Frehse (Eds.): 1st International Workshop
on Synthesis of Continuous Parameters (SynCoP'14).
EPTCS 145, 2014, pp. 3–4, doi:10.4204/EPTCS.145.2

with violation.

The tool Breach [3] provides a versatile framework to perform simulation-based parameter synthesis for PSTL formulas and systems described as ordinary differential equations, Simulink[1] or a generic blackbox simulation function. It has been applyied in industrial contexts in the automative domain [8] and for the analysis of complex biological systems [4, 13, 10].

References

[1] E. Asarin, A. Donzé, O. Maler & D. Nickovic (2011): *Parametric Identification of Temporal Properties*. In: *RV*, pp. 147–160, doi:10.1007/978-3-642-29860-8_12.

[2] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli & M. Weiglhofer (2007): *Specify, Compile, Run: Hardware from PSL*. *Electronic Notes in Theoretical Computer Science* 190(4), pp. 3 – 16, doi:10.1016/j.entcs.2007.09.004. Proceedings of the Workshop on Compiler Optimization meets Compiler Verification (COCV 2007).

[3] A. Donzé (2010): *Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems*. In: *CAV*, pp. 167–170. Available at `http://dx.doi.org/10.1007/978-3-642-14295-6_17`.

[4] A. Donzé, E. Fanchon, L. M. Gattepaille, O. Maler & P. Tracqui (2011): *Robustness Analysis and Behavior Discrimination in Enzymatic Reaction Networks*. *PLoS ONE* 6(9), doi:10.1371/journal.pone.0024246.

[5] A. Donzé & O. Maler (2010): *Robust Satisfaction of Temporal Logic over Real-Valued Signals*. In: *FORMATS*, pp. 92–106, doi:10.1007/978-3-642-14295-6_17.

[6] G. E. Fainekos, A. Girard, H. Kress-Gazit & G. J. Pappas (2009): *Temporal logic motion planning for dynamic robots*. *Automatica* 45(2), pp. 343 – 352, doi:10.1016/j.automatica.2008.08.008. Available at `http://www.sciencedirect.com/science/article/pii/S000510980800455X`.

[7] G.E. Fainekos & G.J. Pappas (2009): *Robustness of Temporal Logic Specifications for Continuous-Time Signals*. *Theoretical Computer Science* 410(42), doi:10.1016/j.tcs.2009.06.021.

[8] X. Jin, A. Donzé, J. Deshmukh & S. A. Seshia (2013): *Mining Requirements from Closed-loop Control Models*. In: *HSCC'13*, doi:10.1145/2461328.2461337.

[9] O. Maler & D. Nickovic (2004): *Monitoring Temporal Properties of Continuous Signals*. In: *FORMATS/FTRTFT*, pp. 152–166, doi:10.1007/978-3-540-30206-3_12.

[10] N. Mobilia, A. Donzé, J.-M. Moulis & E. Fanchon (2012): *A Model of the Cellular Iron Homeostasis Network Using Semi-Formal Methods for Parameter Space Exploration*. In: *HSB*. Available at `http://dx.doi.org/10.4204/EPTCS.92.4`.

[11] P. Nuzzo, H. Xu, N. Ozay, J.B. Finn, A.L. Sangiovanni-Vincentelli, R.M. Murray, A. Donzé & S.A. Seshia (2013): *A Contract-Based Methodology for Aircraft Electric Power System Design*. *Access, IEEE* PP(99), pp. 1–1, doi:10.1109/ACCESS.2013.2295764.

[12] N. Piterman, A. Pnueli & Y. Sa'ar (2006): *Synthesis of Reactive(1) Designs*. In E. Allen Emerson & Kedar S. Namjoshi, editors: *VMCAI*, *Lecture Notes in Computer Science* 3855, Springer, pp. 364–380. Available at `http://dx.doi.org/10.1007/11609773_24`.

[13] S. Stoma, A. Donzé, F. Bertaux, O. Maler & G. Batt (2013): *STL-based Analysis of TRAIL-induced Apoptosis Challenges the Notion of Type I/Type II Cell Line Classification*. *PLoS Comput Biol* 9(5), p. e1003056, doi:10.1371/journal.pcbi.1003056.

[1] `http://www.mathworks.com`

MTL-Model Checking of One-Clock Parametric Timed Automata is Undecidable

Karin Quaas*
Institut für Informatik
Universität Leipzig
D-04109 Leipzig, Germany

Parametric timed automata extend timed automata (Alur and Dill, 1991) in that they allow the specification of *parametric* bounds on the clock values. Since their introduction in 1993 by Alur, Henzinger, and Vardi, it is known that the emptiness problem for parametric timed automata with one clock is decidable, whereas it is undecidable if the automaton uses three or more parametric clocks. The problem is open for parametric timed automata with two parametric clocks. Metric temporal logic, MTL for short, is a widely used specification language for real-time systems. MTL-model checking of timed automata is decidable, no matter how many clocks are used in the timed automaton. In this paper, we prove that MTL-model checking for parametric timed automata is undecidable, even if the automaton uses only one clock and one parameter and is deterministic.

1 Introduction

An important field of algorithmic verification is the analysis of real-time systems, *i.e.*, systems whose behaviour depend on time-critical aspects. Since the early nineties, numerous formalisms have been investigated to express and verify real-time properties. Two prominent examples of such formalisms are *timed automata* and *metric temporal logic*. Timed automata [3] extend classical finite automata with a finite set of real-valued *clocks* whose values grow with the passage of time. The edges of a timed automaton are labelled with *clock constraints* that compare the value of a clock with some constant. An edge can only be taken if the current values of the clocks satisfy the clock constraint labelling the edge. The central property of timed automata is the decidability of the emptiness problem [3].

Metric temporal logic (MTL, for short) extends classical linear temporal logic by constraining the temporal modalities with intervals of the non-negative reals. For example, the formula $\mathsf{F}_{[0,2]}\varphi$ means that φ will hold within two time units from now. Introduced by Koymans in 1990 [17], the satisfiability problem and the model checking problem for timed automata were assumed to be undecidable for a long time. However, more than 20 years later it was proved by Ouaknine and Worrell [19] that both problems are decidable if MTL is interpreted in the pointwise semantics over *finite* timed words. The decidability of the MTL-model checking problem for timed automata is independent of the number of clocks that the timed automaton uses.

A major drawback of timed automata and MTL is that they only allow the specification of *concrete* constraints on timing properties, *i.e.*, one has to provide the concrete values of all time-related constraints that occur in the real-time system. However, it is often more realistic to provide *symbolic* (or, *parametric*) constraints, in particular, if the real-time system under construction is not known in full details in the early stages of design. With the purpose to overcome the incapability of timed automata to express parametric time constraints, *parametric timed automata* were introduced [6]. Parametric timed automata are timed

*The author is supported by Deutsche Forschungsgemeinschaft (DFG), project QU 316/1-1.

É. André and G. Frehse (Eds.): 1st International Workshop
on Synthesis of Continuous Parameters (SynCoP'14).
EPTCS 145, 2014, pp. 5–17, doi:10.4204/EPTCS.145.3

automata defined over a finite set of parameters, which can be used in clock constraints labelling the edges of the automaton. For an example, consider the parametric timed automaton shown in Fig.1 on page 4. The clock y is concretely constrained by a constant like in ordinary timed automata. In contrast to this, the clock x is parametrically constrained by the parameter p. The value of p is determined by a parameter valuation, *i.e.*, a function mapping each parameter to a value in the non-negative reals.

A crucial verification problem for parametric timed automata is the emptiness problem: given a parametric timed automaton $\mathscr{A}$, does there exist some parameter valuation such that $\mathscr{A}$ has an accepting run? However, it turns out that this problem is undecidable already if $\mathscr{A}$ uses three or more parametric clocks [6]. On the positive side, the problem is decidable if in $\mathscr{A}$ at most one clock is compared to parameters. So far nothing is known about the decidability status for parametric timed automata with two parametric clocks; the problem is closely related to some hard and open problems of logic and automata theory [6].

In this paper, we concern ourselves with the MTL-model checking problem for parametric timed automata: given a parametric timed automaton $\mathscr{A}$ and a specification in form of an MTL formula φ, does there exist some parameter valuation such that all finite runs of $\mathscr{A}$ satisfy φ? For parametric timed automata with three clocks, the undecidability of this problem follows from the undecidability of the emptiness problem. Here, we prove that the problem is undecidable even if $\mathscr{A}$ uses only one clock and one parameter and is deterministic. This negative result is in contrast to the decidability of the emptiness problem for one-clock parametric timed automata, and the decidability of MTL-model checking of timed automata. The result can be regarded as further step towards the precise decidability border for the reachability problem for parametric timed automata with two parametric clocks, which is open for more than 20 years.

Related work The reader might wonder why we consider model checking for *parametric* timed automata and *standard* MTL, *i.e.*, a non-parametric extension of MTL. It is well known that if we extend classical LTL with formulae of the form $\varphi_1 \mathsf{U}_{=p} \varphi_2$, meaning that φ_2 has to hold in exactly p steps from now on for some parameter p, then the satisfiability problem ("Given a formula φ, is there some parameter valuation such that φ is satisfiable?") is undecidable: LTL with parameterized *equality modalities* of the form $\mathsf{U}_{=p}$ can be used to encode halting computations of two-counter machines [4]. Undecidablity of the satisfiability problem implies undecidability of the model checking problem for all systems that are capable to recognize the universal language over a given alphabet (as it is the case for, *eg.*, timed automata). In [4] it is also noted that the undecidability proof for LTL with parameterized equality modalities can be adapted to prove the undecidability of the satisfiability problem for LTL extended with parameterized *upper bound modalities* of the form $\mathsf{U}_{\leq p}$ and *lower bound modalities* of the form $\mathsf{U}_{>p}$ unless we restrict every parameter to occur in *either* lower bound modalities *or* upper bound modalities, but not in both.

The restriction on the parameters of a parametric timed automaton to occur either as a lower bound or as an upper bound also forms an important subclass of parametric timed automata, called *lower bound/upper bound (L/U) automata* [15]. For this subclass the emptiness problem is decidable independent of the number of parametric clocks, and for both finite [15] and infinite runs [8]. Model checking L/U automata with parametric extensions of MITL [5] in the *interval-based* semantics is decidable [8, 13]. Recall that constraints occurring at modalities of MITL formulae are not allowed to be of the form $= n$ (not even if the constraint is *concrete, i.e.*, $n \in \mathbb{N}$); in fact, the satisfiability and model checking problems for (non-parametric) MTL in the interval-based semantics are undecidable [14].

A crucial aspect of our undecidability proof is the fact that MTL formulae can be used to encode

computations of *channel machines with insertion errors* [18]: For every channel machine $\mathscr{C}$, there is an MTL formula $\varphi_{\mathscr{C}}$ that is satisfiable if, and only if, $\mathscr{C}$ has a halting computation that may contain insertion errors. This fact was used in [18] to prove the lower complexity bound of the satisfiability problem for MTL over finite timed words. In our proof, we use the parameterized timed automaton to *exclude* insertion errors in the timed words encoding computations of $\mathscr{C}$. We remark that the idea for this proof is similar to the proof of the undecidability for the model checking problem for one-counter machines and Freeze LTL with one register ($\mathrm{LTL}_1^{\downarrow}$, for short) [12]: In [11], it is proved that $\mathrm{LTL}_1^{\downarrow}$ formulae can be used to encode halting computations of *counter automata with incrementing errors*. Like MTL, $\mathrm{LTL}_1^{\downarrow}$ is not capable to exclude such errors. In [12], it is shown that this incapability can be repaired by combining the formula with a non-deterministic one-counter machine. Let us, however, note that there are substantial technical differences between the formalisms MTL and parametric timed automata on the one side, and $\mathrm{LTL}_1^{\downarrow}$ and one-counter machines on the other side.

2 Parametric Timed Automata

We use $\mathbb{N}$, $\mathbb{Q}_{\geq 0}$, and $\mathbb{R}_{\geq 0}$ to denote the non-negative integers, non-negative rationals, and the non-negative reals, respectively. In this section, we fix a finite alphabet Σ, a finite set $\mathscr{P} = \{p_1, \ldots, p_m\}$ of *parameters*, and a finite set $\mathscr{X} = \{x_1, \ldots, x_n\}$ of *clocks*.

We define *clock constraints* ϕ over $\mathscr{X}$ and $\mathscr{P}$ to be conjunctions of formulae of the form $x \sim c$, where $x \in \mathscr{X}$, $c \in \mathbb{N} \cup \mathscr{P}$, and $\sim \in \{<, \leq, =, \geq, >\}$. We use $\Phi(\mathscr{X}, \mathscr{P})$ to denote the set of all clock constraints over $\mathscr{X}$ and $\mathscr{P}$. A *clock valuation* is a function from $\mathscr{X}$ to $\mathbb{R}_{\geq 0}$. For $\delta \in \mathbb{R}_{\geq 0}$, we define $\nu + \delta$ to be $(\nu + \delta)(x) = \nu(x) + \delta$ for each $x \in \mathscr{X}$. For $\lambda \subseteq \mathscr{X}$, we define $\nu[\lambda := 0]$ by $(\nu[\lambda := 0])(x) = 0$ if $x \in \lambda$, and otherwise $(\nu[\lambda := 0])(x) = \nu(x)$.

A parameter valuation is a function $\pi : \mathscr{P} \to \mathbb{Q}_{\geq 0}$ assigning a non-negative rational to each parameter.

A clock valuation ν and a parameter valuation π satisfy a clock constraint ϕ, written $(\nu, \pi) \models \phi$, if the expression obtained from ϕ by replacing each parameter p by $\pi(p)$ and each clock x by $\nu(x)$ evaluates to true.

A *parametric timed automaton* is a tuple $\mathscr{A} = (\Sigma, \mathscr{L}, \mathscr{L}_0, \mathscr{X}, \mathscr{P}, E, \mathscr{L}_F)$, where

- $\mathscr{L}$ is a finite set of locations,
- $\mathscr{L}_0 \subseteq \mathscr{L}$ is the set of *initial* locations,
- $E \subseteq \mathscr{L} \times \Sigma \times \Phi(\mathscr{X}, \mathscr{P}) \times 2^{\mathscr{X}} \times \mathscr{L}$ is a finite set of *edges*,
- $\mathscr{L}_F \subseteq \mathscr{L}$ is the set of *final* locations.

Each edge $(l, a, \phi, \lambda, l')$ represents a discrete transition from l to l' on the input symbol a. The clock constraint ϕ specifies the bounds on the value of the clocks, and the set λ specifies the clocks to be reset to zero.

A *global state* of $\mathscr{A}$ is a pair (l, ν), where $l \in \mathscr{L}$ represents the current location, and the clock valuation ν represents the current values of all clocks. The behaviour of $\mathscr{A}$ depends upon the current global state and the parameter valuation. Each parameter valuation π induces a $(\Sigma, \mathbb{R}_{\geq 0})$-labelled transition relation τ_π over the set of all global states of $\mathscr{A}$ as follows: $\langle (l, \nu), (a, \delta), (l', \nu') \rangle \in \tau_\pi$, where $a \in \Sigma$ and $\delta \in \mathbb{R}_{\geq 0}$, if, and only if, there is an edge $(l, a, \phi, \lambda, l') \in E$ such that for all clocks $x \in \mathscr{X}$ we have $(\nu(x) + \delta, \pi) \models \phi$, and $\nu' = (\nu(x) + \delta)[\lambda := 0]$. A π-run of $\mathscr{A}$ is a finite sequence $\Pi_{1 \leq i \leq k} \langle (l_{i-1}, \nu_{i-1}), (a_i, \delta_i), (l_i, \nu_i) \rangle$ such that $\langle (l_{i-1}, \nu_{i-1}), (a_i, \delta_i), (l_i, \nu_i) \rangle \in \tau_\pi$ for every $i \in \{1, \ldots, k\}$. A π-run is *successful* if $l_0 \in \mathscr{L}_0$, $\nu_0(x) = 0$, and $l_k \in \mathscr{L}_F$.

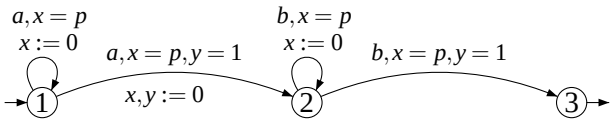


Figure 1: A parametric timed automaton $\mathscr{A}$.

A *timed word* is a non-empty finite sequence $(a_1,t_1)\dots(a_k,t_n) \in (\Sigma \times \mathbb{R}_{\geq 0})^+$ such that the sequence $t_1,\dots,t_n$ of timestamps is non-decreasing. We say that a timed word is *strictly monotonic* if $t_1,\dots,t_n$ is strictly increasing. We use $T\Sigma^+$ to denote the set of finite timed words over Σ. A set $L \subseteq T\Sigma^+$ is called a *timed language*.

Given a parametric timed automaton $\mathscr{A}$ and a parameter valuation π, we associate with each π-run $\Pi_{1\leq i\leq k}\langle(l_{i-1},v_{i-1}),(a_i,\delta_i),(l_i,v_i)\rangle$ the timed word $(a_1,\delta_1)(a_2,\delta_1+\delta_2)\dots(a_k,\sum_{1\leq i\leq k}\delta_k)$. We define $L_\pi(\mathscr{A})$ to be the set of timed words w for which there is a successful π-run of $\mathscr{A}$ that is associated with w. A parameter valuation π is *consistent with* $\mathscr{A}$ if $L_\pi(\mathscr{A})$ is not empty. We use $\Pi(\mathscr{A})$ to denote the set of parameter valuations that are consistent with $\mathscr{A}$.

We say that a parametric timed automaton $\mathscr{A}$ is *deterministic* if $\mathscr{L}_0$ is a singleton, and whenever $(l,a,\phi_1,\lambda_1,l_1)$ and $(l,a,\phi_2,\lambda_2,l_2)$ are two different edges in $\mathscr{A}$, then for all parameter valuations π and clock valuations v we have $(v,\pi) \not\models \phi_1 \wedge \phi_2$.

Example 2.1 *Figure 1 shows a parametric timed automaton over the alphabet $\Sigma = \{a,b\}$ using a parametric clock x and a clock y, and one parameter p. Assume $\pi(p) = n^{-1}$ for some $n \in \mathbb{N}$. Then $L_\pi(\mathscr{A})$ contains a single timed word, namely $(a,\pi(p))(a,2\pi(p))\dots(a,n\pi(p))(b,(n+1)\pi(p))\dots(b,2n\pi(p))$. For all other parameter valuations π, $L_\pi(\mathscr{A}) = \emptyset$, i.e., they are not consistent with $\mathscr{A}$. Hence we have $\Pi(\mathscr{A}) = \{\pi \mid \pi(p) = n^{-1}$ for some $n \in \mathbb{N}\}$. Note that $\mathscr{A}$ is not deterministic, but it can be made deterministic by adding the clock constraint $y < 1$ to the loops in locations 1 and 2.*

3 Metric Temporal Logic

The set of MTL formulae is built up from Σ by boolean connectives and a constraining version of the *until* modality:

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathsf{U}_I \varphi_2$$

where $a \in \Sigma$ and $I \subseteq \mathbb{R}_{\geq 0}$ is an open, closed, or half-open interval with endpoints in $\mathbb{N} \cup \{\infty\}$. Note that we do *not* allow parameters as endpoints. If $I = \mathbb{R}_{\geq 0}$, then we may omit the annotation I on U_I.

We interprete MTL formulae in the *pointwise semantics, i.e.*, over finite timed words over Σ. Let $w = (a_1,t_1)(a_2,t_2)\dots(a_n,t_n)$ be a timed word, and let $i \in \{1,\dots,n\}$. We define the *satisfaction relation for MTL*, denoted by $\models$, inductively as follows:

$$\begin{aligned}
(w,i) \models a &\Leftrightarrow a_i = a \\
(w,i) \models \neg\varphi &\Leftrightarrow (w,i) \not\models \varphi, \\
(w,i) \models \varphi_1 \wedge \varphi_2 &\Leftrightarrow (w,i) \models \varphi_1 \text{ and } (w,i) \models \varphi_2, \\
(w,i) \models \varphi_1 \mathsf{U}_I \varphi_2 &\Leftrightarrow \exists j.i < j \leq |w| : (w,j) \models \varphi_2 \text{ and } t_j - t_i \in I, \text{ and } \forall k.i < k < j : (w,k) \models \varphi_1.
\end{aligned}$$

We say that a timed word $w \in T\Sigma^+$ satisfies an MTL formula φ, written $w \models \varphi$, if $(w,1) \models \varphi$. Given an MTL formula φ, we define $L(\varphi) := \{w \in T\Sigma^+ \mid w \models \varphi\}$. We use the following syntactical abbreviations:

$\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 := \neg\varphi_1 \vee \varphi_2$, $\mathtt{true} := p \vee \neg p$, $\mathtt{false} := \neg\mathtt{true}$, $\mathsf{X}_I\varphi := \mathtt{false}\mathsf{U}_I\varphi$, $\mathsf{F}_I\varphi := \mathtt{true}\mathsf{U}_I\varphi$, $\mathsf{G}_I\varphi := \neg\mathsf{F}_I\neg\varphi$. Observe that the use of the *strict* semantics for the until modality is essential to derive the next modality.

MTL-Model Checking Problem for Parametric Timed Automata

INPUT: A parametric timed automaton $\mathscr{A}$, an MTL formula φ.
QUESTION: Is there some parameter valuation π such that for every $w \in L_\pi(\mathscr{A})$ we have $w \models \varphi$?

In general, the MTL-model checking problem is undecidable for parametric timed automata. This follows from the undecidability of the emptiness problem for parametric timed automata with three or more parametric clocks [6]. In the next section, we prove the undecidability of the MTL-model checking problem for parametric timed automata using one parametric clock and one parameter.

4 Main Result

Theorem 4.1 *The MTL-model checking problem for parametric timed automata is undecidable, even if the automaton uses only one clock and one parameter and is deterministic.*

The remainder of this section is devoted to the proof of Theorem 4.1. The proof is a reduction of the control state reachability problem for channel machines, which we introduce in the following.

4.1 Channel Machines

Let Γ be a finite alphabet. We use ε to denote the *empty word* over Γ. Given two finite words $x, y \in \Gamma^*$, we use $x \cdot y$ to denote the *concatenation* of x any y. We define the order $\leq$ over the set of finite words over Γ by $x_1x_2\ldots x_m \leq y_1y_2\ldots y_n$ if there exists a strictly increasing function $f : \{1,\ldots,m\} \rightarrow \{1,\ldots,n\}$ such that $x_i = y_{f(i)}$ for every $i \in \{1,\ldots,m\}$.

A *channel machine* consists of a finite-state automaton acting on an unbounded fifo channel. Formally, a channel machine is a tuple $\mathscr{C} = (S, s_I, M, \Delta)$, where

- S is a finite set of *control states*,
- $s_I \in S$ is the initial control state,
- M is a finite set of *messages*,
- $\Delta \subseteq S \times L \times S$ is the transition relation over the label set $L = \{m!, m? \mid m \in M\} \cup \{\varepsilon\}$.

A *configuration* of $\mathscr{C}$ is a tuple (s,x), where $s \in S$ is the control state and $x \in M^*$ represents the contents of the channel. The rules in Δ induce an L-labelled transition relation $\rightarrow$ over the set of configurations of $\mathscr{C}$ as follows:

- $\langle (s,x), m!, (s',x') \rangle \in \rightarrow$ if, and only if, there exists some transition $(s, m!, s') \in \Delta$, $x \in \Sigma^*$, and $x' = x \cdot m$, *i.e.*, m is added to the tail of the channel.
- $\langle (s,x), m?, (s',x') \rangle \in \rightarrow$ if, and only if, there exists some transition $(s, m?, s') \in \Delta$, $x' \in \Sigma^*$, and $x = m \cdot x'$, *i.e.*, m is the head of the current channel content.
- $\langle (s,x), \varepsilon, (s',x') \rangle \in \rightarrow$ if, and only if, there exists some transition $(s, \varepsilon, s') \in \Delta$ and $x = \varepsilon$, *i.e.*, the channel is empty, and $x' = x$.

Next, we define another L-labelled transition relation $\rightsquigarrow$ over the set of configurations of $\mathscr{C}$. The relation $\rightsquigarrow$ is a superset of $\rightarrow$. It contains some additional transitions which result from *insertion errors*. We define $\langle (s,x_1),l,(s,x_1')\rangle \in \rightsquigarrow$, if, and only if, $\langle (s,x),l,(s',x')\rangle \in \rightarrow$, $x_1 \leq x$, and $x' \leq x_1'$. A computation of $\mathscr{C}$ is a finite sequence $\Pi_{1\leq i\leq k}\langle (s_{i-1},x_{i-1}),l_i,(s_i,x_i)\rangle$ such that $\langle (s_{i-1},x_{i-1}),l_i,(s_i,x_i)\rangle \in \rightsquigarrow$ for every $i \in \{1,\ldots,k\}$. We say that a computation is *error-free* if for all $i \in \{1,\ldots,k\}$ we have $\langle (s_{i-1},x_{i-1}),l_i,(s_i,x_i)\rangle \in \rightarrow$. Otherwise, we say that the computation is *faulty*.

Control State Reachability Problem for Channel Machines

INPUT: A channel machine $\mathscr{C}$ with control states S, a control state $s_F \in S$.
QUESTION: Is there an error-free computation of $\mathscr{C}$ from (s_I,ε) to (s_F,x) for some $x \in M^*$?

The control state reachability problem is undecidable for channel machines, because channel machines are Turing-powerful [9, 1].

4.2 Encoding Faulty Computations

For the remainder of this section, let $\mathscr{C} = (S,s_I,M,\Delta)$ be a channel machine and let $s_F \in S$. We construct an MTL formula $\varphi_{\mathscr{C}}$ that is satisfiable if, and only if, there exists some $x \in M^*$ such that $\mathscr{C}$ has a computation from (s_I,ε) to (s_F,x) that may be faulty. Later we are going to define a parametric timed automaton $\mathscr{A}_{\mathscr{C}}$ with one clock and one parameter to exclude faulty computations from $L(\varphi_{\mathscr{C}})$.

Let $\Sigma = S \cup M \cup L \cup \{\#,\star\}$, where $\#$ and $\star$ do not occur in $S \cup M \cup L$. We start with defining a timed language $L(\mathscr{C})$ over Σ that consists of all timed words that encode (potentially faulty) computations of $\mathscr{C}$ from (s_I,ε) to (s_F,x) for some $x \in M^*$. The definition of $L(\mathscr{C})$ follows the ideas presented in [18]. Let $\gamma := \Pi_{1\leq i\leq k}\langle (s_{i-1},x_{i-1}),l_i,(s_i,x_i)\rangle$ be a computation of $\mathscr{C}$ with $s_0 = s_I$, $x_0 = \varepsilon$, and $s_k = s_F$. Each configuration (s_i,x_i) occurring in γ is encoded by a timed word of duration one starting with s_0 at time δ for some arbitrary $\delta \in \mathbb{R}_{\geq 0}$. Every symbol s_i is followed by l_{i+1} after one time unit, and by s_{i+1} after two time units. The content x_i of the channel is stored in the time interval between s_i and l_{i+1}. Note that due to the denseness of the time domain we can indeed store the channel content without any restriction on its length. An important detail of the definition of $L(\mathscr{C})$ is that for every message symbol m between s_i and l_{i+1}, there is a copy in the encoding of the next configuration exactly two time units later, unless the label of the current transition is $m?$. In that case, the symbol m is simply removed from the encoding of the configuration.

For our reduction to work, we have to change the idea in some details. First, we define a timed language $L(\mathscr{C},n)$ for every $n \in \mathbb{N}$, where n is non-deterministically chosen and is supposed to represent the expected maximum length of the channel content during a computation. The empty channel in the initial configuration will be represented by a timed word with n hash symbols between s_0 and l_1. Second, we put a stronger condition on the copy policy of the messages. We require that for every hash symbol between s_0 and l_1 there is a message or hash symbol with *the same fractional part* between s_i and l_{i+1} for every $i \in \{1,\ldots,k-1\}$. In Fig. 2, we present some examples to explain the details. (a) If the current instruction is of the form $m_1!$ for some $m_1 \in M$, then in the encoding of the next configuration, the first hash symbol between the control state symbol and the next label symbol is replaced by m_1. (b) If in the encoding of the current configuration there is no hash symbol left, *i.e.*, the expected maximum length of the channel content is exceeded, then a new symbol m_1 is inserted at the end of the encoding of the next configuration. The timestamp of the newly inserted event can be any time strictly between the timestamps of the last message symbol and the next label symbol. (c) If the current instruction is of the form $m_1?$ and the first symbol in the encoding of the current configuration is m_1, then we replace

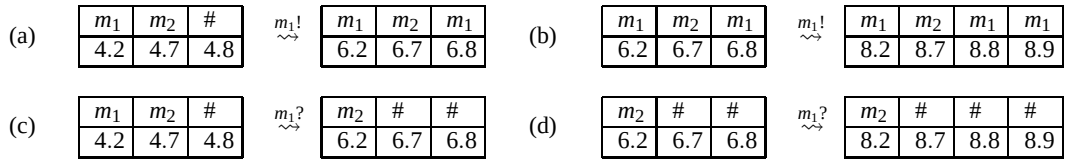


Figure 2: Encoding of the channel content

m_1 by a new hash symbol at the end of the encoding of the next configuration, and additionally shift the fractional parts of the timestamps of the copies of all remaining symbols for one position to the right. (d) If the first symbol is not m_1, *i.e.*, an insertion error is occurring, then we insert a new hash symbol at the end of the encoding of the next configuration. Next, we give the formal definition of $L(\mathscr{C},n)$. Let $n \in \mathbb{N}$. The timed language $L(\mathscr{C},n)$ consists of all timed words w over Σ that satisfy the following conditions:

- w must be strictly monotonic.
- In w, every control state symbol s different from s_F is followed by a label symbol l after one time unit, and by a control state symbol s' after two time units, provided that $(s,l,s') \in \Delta$. The symbol s_F is followed by $\star$ after one time unit. Control state symbols, label symbols and the symbol $\star$ must not occur anywhere else in w.
- Symbols in $M \cup \{\#\}$ may occur in w between a control state symbol and a label symbol. They may not occur anywhere else in w.
- Between a control state and a label symbol, hash symbols # may only occur after message symbols $m \in M$.
- The (untimed) prefix of w must be of the form $s_I \#^n l s$ for some $l \in L, s \in S$.
- w must contain s_F.

Assume that w contains the infix $(s,\delta)(\sigma_1,\delta+\delta_1)(\sigma_2,\delta+\delta_2)\ldots(\sigma_m,\delta+\delta_m)(l,\delta+1)$ for some $s \in S\backslash\{s_F\}$, $l \in L$, $\delta \in \mathbb{R}_{\geq 0}$ and $0 < \delta_1 < \delta_2 < \cdots < \delta_m < 1$.

- If $l = \varepsilon$, then $\sigma_i = \#$ for all $i \in \{1,\ldots,m\}$ (*i.e.*, the channel is indeed empty), and for each σ_i there is a copy two time units later.
- If $l = m!$, then we distinguish between two cases: If there is some $i \in \{1,\ldots,m\}$ such that $\sigma_i = \#$, then *replace* σ_j by m two time units later, where $j \in \{1,\ldots,m\}$ is the smallest number such that $\sigma_j = \#$. For each $k \in \{1,\ldots,m\}\backslash\{j\}$, there is a copy of σ_k two time units later. Otherwise, *i.e.*, if for all $i \in \{1,\ldots,m\}$ we have $\sigma_i \neq \#$, then for each $i \in \{1,\ldots,m\}$, there is a copy of σ_i two time units later. Further, a new symbol m is added between the copy of σ_m and the following symbol in $L \cup \{\star\}$. Note that this corresponds to the case where n has been chosen too small to capture the maximum length of the channel content during the computation.
- If $l = m?$, then we distinguish between two cases: If $\sigma_1 = m$, then for each $i \in \{2,\ldots,m\}$, there is a copy of σ_i two time units after the occurrence of σ_{i-1}. Further there is a new hash symbol two time units after the occurrence of σ_m. Otherwise, *i.e.*, if $\sigma_1 \neq m$, then there is a copy of σ_i two time units later for every $i \in \{1,\ldots,m\}$. Further, the encoding of the next configuration contains an additional hash symbol between the copy of σ_m and the next symbol in $L \cup \{\star\}$. Note that this case corresponds to an *insertion error*.

Let $w_1 = (a_1,t_1)\dots(a_k,t_k)$ and $w_2 = (a'_1,t'_1)\dots(a'_{k'},t'_{k'})$ be two timed words. If $t_k \le t'_1$, then we define the *concatenation* of w_1 and w_2, denoted by $w_1 \cdot w_2$, to be the timed word $(a_1,t_1)\dots(a_k,t_k)(a'_1,t'_1)\dots(a'_{k'},t'_{k'})$. Let $w \in L(\mathscr{C},n)$. We use $\max(w)$ to denote the maximum number of symbols in $M\cup\{\#\}$ that occur in w between a control state symbol and a symbol in $L\cup\{\star\}$. Clearly, every timed word in $L(\mathscr{C},n)$ is of the form

$$(s_0,\delta)\cdot w_1\cdot(l_1,\delta+1)(s_1,\delta+2)\cdot w_2\cdot(l_2,\delta+3)\dots(s_F,\delta+N)\cdot w_N\cdot(\star,\delta+N+1)$$

for some $\delta\in\mathbb{R}_{\ge 0}$ and $N\in\mathbb{N}$, where $s_0 = s_I$ and for every $i\in\{1,\dots,N\}$, w_i is of the form

$$w_i = (\sigma^i_1,\delta+2(i-1)+\delta^i_1)(\sigma^i_2,\delta+2(i-1)+\delta^i_2)\dots(\sigma^i_{n_i},\delta+2(i-1)+\delta^i_{n_i})$$

for some $n_i\in\mathbb{N}$ with $n_1 = n$, and $0<\delta^i_1<\delta^i_2<\dots<\delta^i_{n_i}<1$. In the following, whenever we refer to a timed word $w\in L(\mathscr{C},n)$, we assume that w is of this form. The next lemma states that the fractional parts of the initial time delays $\delta^1_1,\dots,\delta^1_{n_1}$ are not lost. This will be important later.

Lemma 4.2 *Let $n\in\mathbb{N}$ and let $w\in L(\mathscr{C},n)$. For every $i\in\{1,\dots,N-1\}$ there exists a strictly increasing function $f_i:\{1,\dots,n_i\}\to\{1,\dots,n_{i+1}\}$ such that $\delta^i_j = \delta^{i+1}_{f_i(j)}$ for every $j\in\{1,\dots,n_i\}$.*

Proof The proof is by induction on N. (Induction base:) Observe that $\sigma^1_i = \#$ for every $i\in\{1,\dots,n_1\}$. Assume $l_1 = \varepsilon$. Then for every $j\in\{1,\dots,n_1\}$, there is a copy of σ^1_j two time units later. If $l_1 = m!$, then for every $j\in\{2,\dots,n_1\}$, there is a copy of σ^1_j two time units later, and σ^1_1 is replaced by m two time units later. If $l_1 = m?$, then for every $j\in\{1,\dots,n_1\}$, there is a copy of σ^1_j two time units later, and there is an additional symbol # between the copy of $\sigma^1_{n_1}$ and l_2. Whatever case, the definition of $L(\mathscr{C},n)$ does not exclude that new symbols in $M\cup\{\#\}$ are inserted somewhere between s_1 and l_2. Thus we have $n_1\le n_2$. Moreover, since there is a copy for each symbol two time units later, there exists a strictly increasing function $f:\{1,\dots,n_1\}\to\{1,\dots,n_2\}$ such that $\delta^1_j = \delta^2_{f(j)}$ for every $j\in\{1,\dots,n_1\}$. (Induction step) Assume that the claim holds for all $i\in\{1,\dots,k\}$. We prove it also holds for $k+1$. We only treat the two remaining cases. First, assume $l_{k+1} = m?$ and $\sigma^{k+1}_1 = m$. By definition, for every $j\in\{2,\dots,n_{k+1}\}$, there is a copy of σ^{k+1}_j two time units after the occurrence of symbol σ^{k+1}_{j-1}. Further, the first symbol m is replaced by a new hash symbol two time units after the occurrence of $\sigma^{k+1}_{n_{k+1}}$. Second, assume $l_{k+1} = m!$ and we have $\sigma^{k+1}_j\neq\#$ for every $j\in\{1,\dots,n_{k+1}\}$. Then, for each $j\in\{1,\dots,n_{k+1}\}$, there is a copy of σ^{k+1}_j two time units later, and a new symbol m is added after the copy of $\sigma^i_{n_{k+1}}$. Whatever case, the definition of $L(\mathscr{C},n)$ does not exclude that new symbols in $M\cup\{\#\}$ are inserted between s_{k+2} and l_{k+2}. Hence $n_{k+1}\le n_{k+2}$. Since for every $j\in\{1,\dots,n_{k+1}\}$ the symbol σ^{k+1}_j is copied or replaced two time units later, there exists a strictly increasing function $f:\{1,\dots,n_{k+1}\}\to\{1,\dots,n_{k+2}\}$ such that $\delta^{k+1}_j = \delta^{k+2}_{f_{k+1}(j)}$ for every $j\in\{1,\dots,n_{k+1}\}$. □

Let $\gamma := \Pi_{1\le i\le k}\langle(s_{i-1},x_{i-1}),l_i,(s_i,x_i)\rangle$ be a finite computation of $\mathscr{C}$. We use $\max(\gamma)$ to denote the maximum length of the channel content occurring in γ, formally: $\max(\gamma) := \max\{|x_i| \mid 0\le x_i\le k\}$.

Lemma 4.3 *For each error-free computation γ of $\mathscr{C}$ from (s_I,ε) to (s_F,x) for some $x\in M^*$, and every $\delta\in\mathbb{R}_{\ge 0}$, $0<\delta_1<\delta_2<\dots<\delta_{\max(\gamma)}<1$, there exists some timed word $w\in L(\mathscr{C},\max(\gamma))$ such that the prefix of w is of the form $(s_I,\delta)(\#,\delta+\delta_1)\dots(\#,\delta+\delta_{\max(\gamma)})(l_1,\delta+1)$ for some $l_1\in L$, and $\max(w) = \max(\gamma)$.*

Proof Let γ be an errror-free computation of $\mathscr{C}$ of the form $\Pi_{1\le i\le k}\langle(s_{i-1},x_{i-1}),l_i,(s_i,x_i)\rangle$ where $s_0 = s_I$, $x_0 = \varepsilon$ and $s_k = s_F$. Further let $n = \max(\gamma)$. Now assume $\delta\in\mathbb{R}_{\ge 0}$ and $0<\delta_1<\delta_2<\dots<\delta_n<1$. Clearly there is some $w\in L(\mathscr{C},n)$ whose prefix is of the form $u_1 = (s_I,\delta)(\#,\delta+\delta_1)\dots(\#,\delta+\delta_n)(l_1,\delta+1)$. We

prove that there exists some $w \in L(\mathscr{C},n)$ such that u_1 is the prefix of w and $\max(w) = n$, *i.e.*, for every $i \in \{1,\ldots,k\}$, the number of symbols in $M \cup \{\#\}$ between s_{i-1} and l_i (and between s_k and $\star$) is equal to n. The proof is by induction on k.

(Induction base:) Assume $l_1 = \varepsilon$. By definition, there must be a copy for each # exactly two time units later. The addition of new symbols is not required. If $l_1 = m!$, then by definition the first occurrence of # is *replaced* by m exactly two time units later, and for each of the remaining # there is a copy two time units later. The addition of new symbols is not required. Note that the case $m?$ cannot occur because γ is error-free. Hence, there exists some timed word $w \in L(\mathscr{C},n)$ whose prefix is of the form $u_1 \cdot u_2$, where $u_2 = (s_1, 2+\delta)(\sigma_1^2, 2+\delta+\delta_1)(\#, 2+\delta+\delta_2)\ldots(\#, 2+\delta+\delta_n)(l_2, 2+\delta+1)$ for some $\sigma_1^2 \in M \cup \{\#\}$.

(Induction step:) Assume there is some timed word $w \in L(\mathscr{C},n)$ whose prefix is of the form $u_1 \cdot \cdots \cdot u_p$ for some $p < k$, where for every $i \in \{1,\ldots,p\}$, u_i is of the form

$$(s_{i-1}, 2(i-1)+\delta)(\sigma_1^i, 2(i-1)+\delta+\delta_1)(\sigma_2^i, 2(i-1)+\delta+\delta_2)\ldots(\sigma_n^i, 2(i-1)+\delta+\delta_n)(l_i, 2(i-1)+\delta+1)$$

for some $\sigma_1^i,\ldots,\sigma_n^i \in M \cup \{\#\}$.

Assume $l_p = m?$ for some $m \in M$. By the fact that γ is error-free, we know $\sigma_1^p = m$. By definition, there is a copy of σ_i^p two time units after the occurrence of σ_{i-1}^p for every $i \in \{2,\ldots,n\}$, and there is a new hash symbol inserted two time units after the occurrence of σ_n^p. The addition of new symbols is not required.

Assume $l_p = m!$ for some $m \in M$. Recall that $n = \max(\gamma)$ is the maximum length of the channel content in γ. Hence there must be some $j \in \{1,\ldots,n\}$ such that $\sigma_j^p = \#$. By definition, the smallest $j \in \{1,\ldots,n\}$ with $\sigma_j^p = \#$ is replaced by m exactly two time units later. For each of the remaining symbols there is a copy two time units later. The addition of new symbols is not required.

Assume $l_p = \varepsilon$. We can proceed as above, concluding that the addition of new symbols is not required.

Hence, there exists some timed word $w \in L(\mathscr{C},n)$ whose prefix is of the form $u_1 \cdot u_2 \cdot \ldots u_p \cdot u_{p+1}$, where $u_{p+1} = (s_p, 2p+\delta)(\sigma_1^{p+1}, 2p+\delta+\delta_1)(\sigma_2^{p+1}, 2p+\delta+\delta_2)\ldots(\sigma_n^{p+1}, 2p+\delta+\delta_n)(l_{p+1}, 2p+\delta+1)$ for some $\sigma_1^{p+1},\ldots,\sigma_n^{p+1} \in M \cup \{\#\}$.

We thus have proved that there exists some $w \in L(\mathscr{C},n)$ with $\max(w) = n$. □

Lemma 4.4 *For each $n \in \mathbb{N}$ and $w \in L(\mathscr{C},n)$ with $\max(w) = n$, there exists some error-free computation γ of $\mathscr{C}$ from (s_I,ε) to (s_F,x) for some $x \in M^*$ with $\max(\gamma) \leq n$.*

Proof Let $n \in \mathbb{N}$ and let $w \in L(\mathscr{C},n)$ such that $\max(w) = n$. Hence the number of symbols in $M \cup \{\#\}$ between every control state symbol and the following label symbol (or the symbol $\star$ if the state symbol is s_F) in w is constantly equal to n. This implies that (1) whenever a control state symbol s is followed by a label symbol $m?$ one time unit later, then the next symbol after s must be m, which will be replaced by a new hash symbol; (2) whenever a state symbol s is followed by a label symbol $m!$ one time unit later, then there must exist some hash symbol in between, and the first such hash symbol will be replaced by m; and (3) w does not contain any spontaneously inserted symbols. From (1) and (3) we can conclude that w encodes an error-free computation. From (2) we can conclude that the choice of n is big enough to capture the maximum length of the channel content. Hence there exists some error-free computation of $\mathscr{C}$ from (s_I,ε) to (s_F,x) for some $x \in M^*$ with $\max(\gamma) \leq n$. □

4.3 Excluding Faulty Computations

Next we define a parametric timed automaton $\mathscr{A}_{\mathscr{C}}$ over $\Sigma_{\mathscr{C}}$ such that $L(\mathscr{C},n) \cap L(\mathscr{A}_{\mathscr{C}})$ consists of all timed words that encode *error-free* computations of $\mathscr{C}$ from (s_I,ε) to (s_F,x) for some $x \in M^*$. The

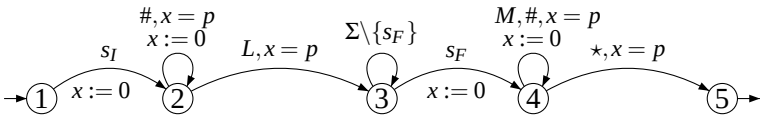


Figure 3: The parametric timed automaton $\mathscr{A}_{\mathscr{C}}$ that excludes insertion errors.

parametric timed automaton $\mathscr{A}_{\mathscr{C}}$ is shown in Fig. 3. It uses one clock x, parametrically constrained by a single parameter p. Note that $\mathscr{A}_{\mathscr{C}}$ is deterministic.

Theorem 4.5 *$\mathscr{C}$ has an error-free computation from (s_0,ε) to (s_F,x) for some $x \in M^*$, if, and only if, there exist $n \in \mathbb{N}$ and a parameter valuation π such that $L(\mathscr{C},n) \cap L_\pi(\mathscr{A}_{\mathscr{C}}) \neq \emptyset$.*

Proof For the direction from left to right, let $\gamma := \Pi_{1\le i\le k}\langle (s_{i-1},x_{i-1}), l_i, (s_i,x_i)\rangle$ be an error-free computation of $\mathscr{C}$ such that $s_0 = s_I$, $x_0 = \varepsilon$ and $s_k = s_F$. Define $n = \max(\gamma)$. Let $\delta \in \mathbb{R}_{\ge 0}$, and define $\delta_i = \frac{i}{(n+1)}$ for every $i \in \{1,\dots,n\}$. By Lemma 4.3, there exists $w \in L(\mathscr{C},n)$ such that the prefix of w is of the form

$$(s_I,\delta)(\#,\delta+\delta_1)\dots(\#,\delta+\delta_n)(l_1,\delta+1)$$

and $\max(w) = n$. This together with Lemma 4.2 implies that the suffix of w is of the form

$$(s_F,2k+\delta)(\sigma_1,2k+\delta+\delta_1)\dots(\sigma_n,2k+\delta+\delta_n)(\star,2k+\delta+1)$$

for some $\sigma_1,\dots,\sigma_n \in M \cup \{\#\}$. Note that in both the prefix and the suffix of w the time delay between every symbol is δ_1. Define $\pi(p) = \delta_1$. It is easy to see that $w \in L_\pi(\mathscr{A}_{\mathscr{C}})$. Hence $L(\mathscr{C},n) \cap L_\pi(\mathscr{A}_{\mathscr{C}}) \neq \emptyset$.

For the direction from right to left, assume there exist $n \in \mathbb{N}$ and a parameter valuation π such that $L(\mathscr{C},n) \cap L_\pi(\mathscr{A}_{\mathscr{C}}) \neq \emptyset$. Let $w \in L(\mathscr{C},n) \cap L_\pi(\mathscr{A}_{\mathscr{C}})$. By definition of $L(\mathscr{C},n)$, the prefix of w is of the form

$$(s_I,\delta)(\#,\delta+\delta_1)(\#,\delta+\delta_2)\dots(\#,\delta+\delta_n)(l,\delta+1)$$

for some $\delta \in \mathbb{R}_{\ge 0}$, $0 < \delta_1 < \delta_2 < \dots < \delta_n < 1$, and $l \in L$. The clock constraints at the loop in location 2 and at the edge from location 2 to 3 implies $\delta_i = \frac{i}{(n+1)}$ for every $i \in \{1,\dots,n\}$ and $\pi(p) = \delta_1$. By Lemma 4.2, the suffix of w must be of the form

$$(s_F,N+\delta)(\sigma_n,N+\delta+\delta'_1)\dots(\sigma_m,N+\delta+\delta'_m)(\star,N+\delta+1)$$

for some $N \in \mathbb{N}$, $0 < \delta'_1 < \delta'_2 < \dots < \delta'_m < 1$ such that $n \le m$, and there exists a strictly increasing function $f : \{1,\dots,n\} \to \{1,\dots,m\}$ such that $\delta_i = \delta'_{f(i)}$. Note that $\star$ occurs exactly one time unit after s_F. This, together with the clock constraints at the loop in location 4 and at the edge from 4 to the final location 5, implies $m = n$ (and $\delta'_i = \delta_i$ for every $i \in \{1,\dots,n\}$). By Lemma 4.2, we further know that the number of symbols between a control state symbol and a symbol in $L \cup \{\star\}$ cannot *decrease*, and hence it follows that $\max(w) = n$. By Lemma 4.4, there exists an error-free computation of $\mathscr{C}$ from (s_0,ε) to (s_F,x) for some $x \in M^*$. □

4.4 The Reduction

We define $L(\mathscr{C}) = \cup_{n\in\mathbb{N}} L(\mathscr{C},n)$. Then we obtain

Corollary 4.6 *There exists an error-free computation of $\mathscr{C}$ from (s_I,ε) to (s_F,x) for some $x \in M^*$, if, and only if, there exists some parameter valuation π with $L_\pi(\mathscr{A}_{\mathscr{C}}) \cap L(\mathscr{C}) \neq \emptyset$.*

Next, we define the MTL formula $\varphi_{\mathscr{C}}$ such that $L(\varphi_{\mathscr{C}}) = L(\mathscr{C})$. The formula $\varphi_{\mathscr{C}}$ is the conjunction of a set of formulas, each of them expressing one of the conditions of $L(\mathscr{C})$. We start by defining some auxiliary formulas: $\bigvee S := \bigvee_{s\in S} s$, $\bigvee M := \bigvee_{m\in M} m$, $\bigvee L := \bigvee_{l\in L} l$, $\varphi_{\mathsf{copyM}} := \mathsf{G}_{(0,1)} \bigwedge_{m\in M}(m \to \mathsf{F}_{=2} m)$, and $\varphi_{\mathsf{copy\#}} := \mathsf{G}_{(0,1)}(\# \to \mathsf{F}_{=2}\#)$.

- $\mathsf{G}(\mathsf{X}_{>0}\mathtt{true} \vee \neg \mathsf{X}\mathtt{true})$ (Strict monotonicity)
- $\mathsf{G}\langle \bigwedge_{s\in S\setminus\{s_F\}}(s \to \bigvee_{(s,l,s)\in\Delta}(\mathsf{F}_{=1} l \wedge \mathsf{F}_{=2} s')) \wedge (s_F \to \mathsf{F}_{=1}\star)\rangle$, $\mathsf{G}\langle \bigvee S \to ((\mathsf{G}_{<2}\neg \bigvee S) \wedge (\mathsf{G}_{(0,1)\cup(1,2)} \neg \bigvee L)\rangle$ (Conditions on the occurrence of control state symbols and symbols in $L \cup \{\star\}$)
- $\mathsf{G}\langle \bigvee S \to (\mathsf{G}_{(0,1)}(\bigvee M \vee \#) \wedge \mathsf{G}_{[1,2)} \neg(\bigvee M \vee \#))\rangle$, $\mathsf{G}((\# \wedge \mathsf{X} \bigvee M) \to \mathtt{false})$ (Conditions on symbols in $M \cup \{\#\}$)
- $s_I \wedge \bigvee_{(s_I,l,s)\in\Delta}(\# \mathsf{U}(l \wedge \mathsf{X}s))$ (Encoding of the initial configuration)
- $\mathsf{F}s_F$ (Reaching s_F)
- $\mathsf{G} \bigwedge_{\substack{(s,\varepsilon,-)\in\Delta \\ s\neq s_F}} ((s \wedge \mathsf{F}_{=1}\varepsilon) \to ((\mathsf{G}_{(0,1)} \neg \bigvee M) \wedge \varphi_{\mathsf{copy\#}}))$
- $\mathsf{G} \bigwedge_{\substack{\delta=(s,m!,-)\in\Delta \\ s\neq s_F}} ((s \wedge \mathsf{F}_{=1} m!) \to (\varphi_{\mathsf{copyM}} \wedge \varphi_{next\#} \wedge \varphi_{yes\#} \wedge \varphi_{no\#}))$, where
  - $\varphi_{next\#} = \mathsf{X}\# \to (\mathsf{X}\mathsf{F}_{=2} m \wedge \mathsf{X}\varphi_{\mathsf{copy\#}})$
  - $\varphi_{yes\#} = (\mathsf{F}_{<1} \wedge \neg \mathsf{X}\#)\# \to \mathsf{G}_{<1}((\neg\# \wedge \mathsf{X}\#) \to \mathsf{X}\mathsf{F}_{=2} m \wedge \mathsf{X}\varphi_{\mathsf{copy\#}}))$
  - $\varphi_{no\#} = \neg \mathsf{F}_{<1}\# \to \mathsf{G}_{<1}(\mathsf{X}m! \to \mathsf{F}_{=2}(\mathsf{X}m \wedge \mathsf{X}\mathsf{X} \bigvee L))$
- $\mathsf{G} \bigwedge_{\substack{(s,m?,-)\in\Delta \\ s\neq s_F}} ((s \wedge \mathsf{F}_{=1} m?) \to (\varphi_{\mathsf{yesm}} \wedge \varphi_{\mathsf{nom}}))$, where
  - $\varphi_{\mathsf{yesm}} = \mathsf{X}m \to (\varphi_{\mathsf{shift}} \mathsf{U} m?)$, $\varphi_{\mathsf{shift}} = \bigwedge_{m\in M}(\mathsf{X}m \to \mathsf{F}_{=2} m) \wedge (\mathsf{X}\# \to \mathsf{F}_{=2}\#) \wedge (\mathsf{X}m? \to \mathsf{F}_{=2}\#)$
  - $\varphi_{\mathsf{nom}} = \mathsf{X}\neg m \to (\varphi_{\mathsf{copyM}} \wedge \varphi_{\mathsf{copy\#}} \wedge \mathsf{G}_{<1}(\mathsf{X}m? \to \mathsf{F}_{=2}(\mathsf{X}\# \wedge \mathsf{X}\mathsf{X} \bigvee L)))$

Proof of Theorem 4.1 Let $\mathscr{C} = (S, s_0, M, \Delta)$ be a channel machine, let $s_F \in S$. Define the parametric timed automaton $\mathscr{A}_{\mathscr{C}}$ and the MTL formula $\varphi_{\mathscr{C}}$ as above. By Corollary 4.6 we know that there is an error-free computation from (s_I, ε) to (s_F, x) for some $x \in M^*$, if, and only if, there exists some parameter valuation π with $L_\pi(\mathscr{A}_{\mathscr{C}}) \cap L(\varphi_{\mathscr{C}}) \neq \emptyset$. The latter, however, is equivalent to $L_\pi(\mathscr{A}_{\mathscr{C}}) \not\subseteq L(\neg\varphi_{\mathscr{C}})$, *i.e.*, there exists some timed word $w \in L_\pi(\mathscr{A}_{\mathscr{C}})$ such that $w \not\models \neg\varphi_{\mathscr{C}}$. Hence, the MTL-model checking problem for parametric timed automata is undecidable. □

5 Discussion

For our undecidability result we construct a parametric timed automaton using a parametric *equality* constraint of the form $x = p$. Parametric equality constraints seem to be a source of undecidability; they occur in the undecidability proofs of, *eg.*, the emptiness problem for parametric timed automata with three clocks [6], and the satisfiability problem for a parametric extension of LTL [4]. A natural question is thus to consider the MTL-model checking problem for L/U-automata [15], a subclass of parametric timed automata in which parameters are only allowed to occur either as a lower bound or as an upper bound, but not both, and for which the emptiness problem is decidable independent of the number of clocks. We further remark that the proof does not work if we restrict the parameter valuation to be a function mapping each parameter to a non-negative integer.

Acknowledgements I would like to thank James Worrell for pointing me to MTL's capability to encode computations of *Turing machines with insertion errors*, explained in [18].

References

[1] Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, Karin Quaas & James Worrell (2008): *Universality Analysis for One-Clock Timed Automata*. *Fundam. Inform.* 89(4), pp. 419–450. Available at `http://iospress.metapress.com/content/xx63231v71037607/`.

[2] Luca Aceto & Anna Ingólfsdóttir, editors (2006): *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25-31, 2006, Proceedings*. *Lecture Notes in Computer Science* 3921, Springer.

[3] Rajeev Alur & David L. Dill (1994): *A Theory of Timed automata*. *Theor. Comput. Sci.* 126(2), pp. 183–235. Available at `http://dx.doi.org/10.1016/0304-3975(94)90010-8`.

[4] Rajeev Alur, Kousha Etessami, Salvatore La Torre & Doron Peled (2001): *Parametric temporal logic for "model measuring"*. *ACM Trans. Comput. Log.* 2(3), pp. 388–407. Available at `http://doi.acm.org/10.1145/377978.377990`.

[5] Rajeev Alur, Tomás Feder & Thomas A. Henzinger (1996): *The Benefits of Relaxing Punctuality*. *J. ACM* 43(1), pp. 116–146. Available at `http://doi.acm.org/10.1145/227595.227602`.

[6] Rajeev Alur, Thomas A. Henzinger & Moshe Y. Vardi (1993): *Parametric real-time reasoning*. In Kosaraju et al. [16], pp. 592–601. Available at `http://doi.acm.org/10.1145/167088.167242`.

[7] Roberto M. Amadio, editor (2008): *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*. *Lecture Notes in Computer Science* 4962, Springer.

[8] Laura Bozzelli & Salvatore La Torre (2009): *Decision problems for lower/upper bound parametric timed automata*. *Formal Methods in System Design* 35(2), pp. 121–151. Available at `http://dx.doi.org/10.1007/s10703-009-0074-0`.

[9] Daniel Brand & Pitro Zafiropulo (1983): *On Communicating Finite-State Machines*. *J. ACM* 30(2), pp. 323–342, Available at `http://doi.acm.org/10.1145/322374.322380`.

[10] Adrian Horia Dediu, Henning Fernau & Carlos Martín-Vide, editors (2010): *Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings*. *Lecture Notes in Computer Science* 6031, Springer. Available at `http://dx.doi.org/10.1007/978-3-642-13089-2`.

[11] Stéphane Demri & Ranko Lazić (2009): *LTL with the freeze quantifier and register automata*. *ACM Trans. Comput. Log.* 10(3). Available at `http://doi.acm.org/10.1145/1507244.1507246`.

[12] Stéphane Demri, Ranko Lazić & Arnaud Sangnier (2008): *Model Checking Freeze LTL over One-Counter Automata*. In Amadio [7], pp. 490–504. Available at `http://dx.doi.org/10.1007/978-3-540-78499-9_34`.

[13] Barbara Di Giampaolo, Salvatore La Torre & Margherita Napoli (2010): *Parametric Metric Interval Temporal Logic*. In Dediu et al. [10], pp. 249–260. Available at `http://dx.doi.org/10.1007/978-3-642-13089-2_21`.

[14] Thomas Henzinger (1991): *The temporal specification and verification of real-time systems.* Ph.D. thesis, Stanford University. Technical Report STAN-CS-91-1380.

[15] Thomas Hune, Judi Romijn, Mariëlle Stoelinga & Frits W. Vaandrager (2002): *Linear parametric model checking of timed automata*. *J. Log. Algebr. Program.* 52-53, pp. 183–220. Available at `http://dx.doi.org/10.1016/S1567-8326(02)00037-1`.

[16] S. Rao Kosaraju, David S. Johnson & Alok Aggarwal, editors (1993): *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*. ACM.

[17] Ron Koymans (1990): *Specifying Real-Time Properties with Metric Temporal Logic*. *Real-Time Systems* 2(4), pp. 255–299. Available at `http://dx.doi.org/10.1007/BF01995674`.

[18] Joël Ouaknine & James Worrell (2006): *On Metric Temporal Logic and Faulty Turing Machines*. In Aceto & Ingólfsdóttir [2], pp. 217–230. Available at `http://dx.doi.org/10.1007/11690634_15`.

[19] Joël Ouaknine & James Worrell (2007): *On the decidability and complexity of Metric Temporal Logic over finite words*. *Logical Methods in Computer Science* 3(1). Available at `http://dx.doi.org/10.2168/LMCS-3(1:8)2007`.

Probabilistic Bisimulations for PCTL Model Checking of Interval MDPs

Vahid Hashemi
MPI für Informatik
Saarbrücken, Germany
Department of Computer Science
Saarland University
Saarbrücken, Germany
hashemi@mpi-inf.mpg.de

Hassan Hatefi
Department of Computer Science
Saarland University
Saarbrücken, Germany
hhatefi@cs.uni-saarland.de

Jan Krčál
Department of Computer Science
Saarland University
Saarbrücken, Germany
krcal@cs.uni-saarland.de

Verification of PCTL properties of MDPs with convex uncertainties has been investigated recently by Puggelli et al. However, model checking algorithms typically suffer from state space explosion. In this paper, we address probabilistic bisimulation to reduce the size of such an MDPs while preserving PCTL properties it satisfies. We discuss different interpretations of uncertainty in the models which are studied in the literature and that result in two different definitions of bisimulations. We give algorithms to compute the quotients of these bisimulations in time polynomial in the size of the model and exponential in the uncertain branching. Finally, we show by a case study that large models in practice can have small branching and that a substantial state space reduction can be achieved by our approach.

1 Introduction

Modelling formalisms like Markov decision processes (MDP) [29] or Probabilistic automata (PA) [31] are used for representing systems that combine non-deterministic and probabilistic behaviour. They can be viewed as transition systems where in each step an outgoing transition of the current state is chosen *non-deterministically* and the successor state is chosen *randomly* according to a fixed probability distribution assigned to this transition. Assigning fixed probability distributions to transitions is however not realistic [18,22] in many modelling scenarios: measurement errors, statistical estimates, or mathematical approximations all lead to *intervals* instead of fixed probabilities.

Interval MDPs [28] (also called *Bounded-parameter MDPs* [13, 37]) address this need by bounding the probabilities of each successor state by an interval instead of a fixed number. In such a model, the transition probabilities are not fully specified and this uncertainty again needs to be resolved non-deterministically. The two sources of non-determinism have *different* interpretation in different applications:

1. In verification of parallel systems with uncertain transition probabilities [28] the transitions correspond to unpredictable interleaving of computation of the communicating agents. Hence, both the choice of transitions and their probability distributions is *adversarial*.
2. In control synthesis for systems with uncertain probabilities [36] the transitions correspond to various control actions. We search for a choice of transitions that is *optimal* against an adversarial choice of probability distributions satisfying the interval bounds.
3. In parameter synthesis for parallel systems [14] the transition probabilities are underspecified to allow freedom in implementation of such a model. We search for a choice of probability dis-

É. André and G. Frehse (Eds.): 1st International Workshop
on Synthesis of Continuous Parameters (SynCoP'14).
EPTCS 145, 2014, pp. 19–33, doi:10.4204/EPTCS.145.4

tributions that is optimal for adversarial choice of transitions (again stemming from the possible interleaving).

Furthermore, the choice of probability distributions satisfying the interval constraints can be either resolved statically [18], i.e. at the beginning once for all, or dynamically [17, 33], i.e. independently for each computation step. Here, we focus on the dynamic approach that is easier to work with algorithmically and can be seen as a relaxation of the static approach that is often intractable [2, 7, 11, 33].

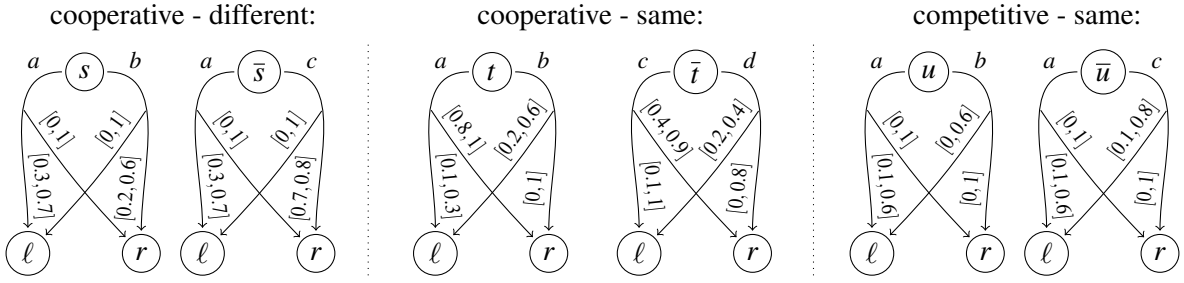
There are several algorithms [28, 36] to check whether a given interval MDP satisfies a given specification expressed in a *logic* like PCTL [15] or LTL [27]. However, models often suffer from state space explosion when obtained using some higher-level modelling formalism such as a process algebra. These models usually contain redundancy that can be removed without changing the behaviour of the model. One way to reason about such behavioural equivalence is *bisimulation* [24]. For a given huge model it allows to construct the *bisimulation quotient*, the smallest model with equivalent behaviour – in particular preserving all its properties expressible by a logic such as PCTL.

Our contribution In this paper, we define the first bisimulations for interval MDPs (that are also the first bisimulations for MDPs with uncertain transitions in general). We show that different interpretation of non-determinism yields two different bisimulations: one for models where the two non-determinisms are resolved in a *cooperative way* (see point 1. above), another for models where it is resolved in a *competitive way* (see points 2. and 3. above).

Furthermore, we show how to compute these bisimulations by algorithms based on comparing polytopes of probability distributions associated with each transition. The algorithms are fixed parameter tractable with respect to the maximal dimension of the polytopes (i.e. maximal number of different states that an uncertain transition can lead to); in the competitive case also with respect to the maximal number of outgoing uncertain transitions. Note that in many applications these parameters are small.

We finally argue by a case study that, if uncertainty stems from a small number of different phenomena such as *node failure* or *loss of a message*, the same shape of polytopes will repeat many times over the states space. We demonstrate that the redundancy in this case may result in a massive state space minimisation.

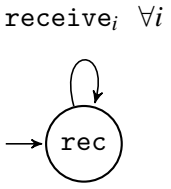
Example 1.1. We illustrate the contribution by two examples. In the first one, we explain how the competitive and the cooperative resolution of non-determinism result in different behavioural equivalences. Consider the three pair of states below.



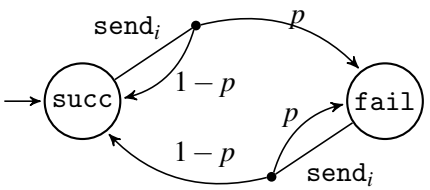
As regards the cooperative non-determinism, s has not the same behaviour as $\bar{s}$ since $\bar{s}$ can move to r with probability 0.8 by choosing c and $(\ell \mapsto 0.2, r \mapsto 0.8)$, which s cannot simulate. So far the equivalence might seem easy to check. However, note that t has the same behaviour as $\bar{t}$ even though the interval bounds for the transitions quite differ. Indeed, the sets of distributions satisfying the interval constraints are the same for t and $\bar{t}$.

As regards the competitive non-determinism, observe that u and $\overline{u}$ have also the same behaviour. Indeed, the a transitions coincide and both b and c offer a wider choice of probability distributions than a. If the most adversarial choice of the distribution scheduler lies in the difference $[b] \setminus [a]$ of the distributions offered by b and a, the transition scheduler then never chooses b; hence a in $\overline{u}$ can simulate both a and b in u. In the other direction it is similar and u and $\overline{u}$ have the same behaviour although $[b] \neq [c]$.

Example 1.2. In the second example, we explain the redundancy of large models with a small source of uncertainty. Consider a Wireless Sensor Network (WSN) containing N sensors $S_1, S_2 \cdots S_N$ and a gateway G, all communicating over an unreliable channel. For simplicity, we assume that each sensor continuously sends some data to the gateway which are then pushed into an external server for further analysis. As the channel is unreliable, with some positive probability p each message with data may get lost. The WSN can be seen as the parallel composition of gateway G and sensors S_i depicted below that synchronise over labels $\texttt{send}_i$'s and $\texttt{receive}_i$'s.



(a) Gateway G



(b) Sensor S_i

For instance environmental effects on radio transmission, mobility of sensor nodes or traffic burst (see e. g. [30]) cause that the exact probability of failure is unknown. The estimation of this probability, e.g. by empirical data analysis, usually leads to an interval $p \in [\ell, u]$ which turns the model into an interval MDP.

Let us stress that there is only one source of uncertainty appearing all over the state space no matter what is the number of sensors N. This makes many states of the model behave similarly. For example in the WSN, the parallel composition of the above model has 2^N states. However one can show that the bisimulation quotient has only $N+1$ states. Indeed, all states that have the same number of failed sensors have the same behaviour. Thus, for limited source of uncertainty in a model obtained by compositional modelling, the state space reduction may be enormous.

Related work Various probabilistic modelling formalisms with uncertain transitions are studied in the literature. Interval Markov chains [18, 22] or Abstract Markov chains [12] extend standard discrete-time Markov chains (MC) with interval uncertainties and thus do not feature the non-deterministic choice of transitions. Uncertain MDPs [25, 28, 36] allow more general sets of distributions to be associated with each transition, not only those described by intervals. Usually, they restrict to *rectangular uncertainty sets* requiring that the uncertainty is linear and independent for any two transitions of any two states. Our general algorithm working with polytopes can be easily adapted to this setting. Parametric MDPs [14] to the contrary allow such dependencies as every probability is described as a rational function of a finite set of global parameters.

From the side of view of compositional specification, Interval Markov chains [18] and Abstract probabilistic automata [9,10] serve as specification theories for MC and PA featuring satisfaction relation, and various refinement relations. In order to be closed under parallel composition, Abstract PA allow

general polynomial constraints on probabilities instead of interval bounds. Since for Interval MC it is not possible to explicitly construct parallel composition, the problem whether there is a common implementation of a set of Interval Markov chains is addressed instead [11]. To the contrary, interval bounds on *rates* of outgoing transitions work well with parallel composition in the continuous-time setting of Abstract interactive Markov chains [20]. The reason is that unlike probabilities, the rates do not need to sum up to 1. A different way [38] to successfully define parallel composition for interval models is to separate synchronising transitions from the transitions with uncertain probabilities. This is also the core of our approach to parallel composition when constructing a case study as discussed in Section 5.

We are not aware of any existing bisimulation for uncertain or parametric probabilistic models. Among similar concepts studied in the literature are simulation [38] and refinement [9, 11, 18] relations for previously mentioned models. Our definition of bisimulation in the competitive setting is inspired by the alternating bisimulation [1, 6].

Many new verification algorithms for interval models appeared in last few years. Reachability and expected total reward is addressed for for Interval MC [8] as well as Interval MDP [37]. PCTL model checking and LTL model checking are studied for Interval MC [2,7,8] and also for Interval MDP [28,36]. Among other technical tools, all these approaches make use of (robust) dynamic programming relying on the fact that transition probability distributions are resolved dynamically. For the static resolution of distributions, adaptive discretisation technique for PCTL parameter synthesis is given in [14]. Uncertain models are also widely studied in the control community [13, 25, 37], mainly interested in maximal expected finite-horizon reward or maximal expected discounted reward.

Structure of the paper We start with necessary preliminaries in Section 2. In Section 3, we give the definitions of probabilistic bisimulations for interval MDP and discuss their properties and differences. In Section 4, we give the FPT algorithms for both cooperative and competitive cases. Finally, in Section 5 we demonstrate our approach on a case study. Due to space limitations, we refer the reader interested in detailed proofs to [16].

2 Preliminaries

In this paper, the sets of all positive integers, rational numbers, real numbers and non-negative real numbers are denoted by $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{R}^{\geq 0}$, respectively. For a set X, we denote by $\Delta(X)$ the set of discrete probability distributions over X.

2.1 Interval Markov Decision Processes

Let us formally define Interval *MDP*.

Definition 1 (*IMDP*)**.** *An* Interval Markov Decision Process *(IMDP)* $\mathfrak{M}$ *is a tuple* (S, A, AP, L, I)*, where* S *is a finite set of* states*,* A *is a finite set of* actions*,* AP *is a finite set of* atomic propositions*,* $L\colon S \to 2^{AP}$ *is a* labelling function*, and* $I\colon S \times A \times S \to \mathbb{I}$ *is an* interval transition probability function *where* $\mathbb{I}$ *is a set of subintervals of* $[0,1]$.

Furthermore, for each state s and action a, we denote by $s \xrightarrow{a} \mu$ that $\mu \in \Delta(S)$ is a *feasible distribution*, i.e. for each state s' we have $\mu(s') \in I(s,a,s')$. We require that the set $\{\mu \mid s \xrightarrow{a} \mu\}$, also denoted by $\mathscr{C}^{s,a}$, is non-empty for each state s and action a.

An interval MDP is initiated in some state s_1 and then moves in discrete steps from state to state forming an infinite path $s_1 s_2 s_3 \cdots$. One step, say from state s_i, is performed as follows. First, an action $a \in A$ is chosen non-deterministically by *Scheduler*. Then, *Nature* resolves the uncertainty and chooses non-deterministically one corresponding feasible distribution $\mu \in \mathscr{E}^{s_i,a}$. Finally, the next state s_{i+1} is chosen randomly according to the distribution μ.

Let us define the semantics of an *IMDP* formally. A *path* is a finite or infinite sequence of states $\omega = s_1 s_2 \cdots$. For a finite path ω, we denote by $last(\omega)$ the last state of ω. The set of all finite paths and the set of all infinite paths are denoted by $Paths_{fin}$ and $Paths_{inf}$, respectively. Furthermore, let $Paths_\omega = \{\omega\omega' \mid \omega' \in Paths_{inf}\}$ denote the set of paths that have the finite prefix $\omega \in Paths_{fin}$.

Definition 2 (*Scheduler* and *Nature*)**.** *A* scheduler *is a function $\sigma : Paths_{fin} \to \Delta(A)$ that to each finite path ω assigns a distribution over the set of actions. A* nature *is a function $\pi : Paths_{fin} \times A \to \Delta(S)$ that to each finite path ω and action a assigns a feasible distribution, i.e. an element of $\mathscr{E}^{s,a}$ where $s = last(\omega)$. We denote by Σ the set of all schedulers and by Π the set of all natures.*

For an initial state s, a scheduler σ, and a nature π, let $\Pr_s^{\sigma,\pi}$ denote the unique probability measure over $(Paths_{inf}, \mathscr{B})$[1] such that the probability $\Pr_s^{\sigma,\pi}[Paths_{s'}]$ of starting in s' equals 1 if $s = s'$ and 0, otherwise; and the probability $\Pr_s^{\sigma,\pi}[Paths_{\omega s'}]$ of traversing a finite path $\omega s'$ equals $\Pr_s^{\sigma,\pi}[Paths_\omega] \cdot \sum_{a \in A} \sigma(\omega)(a) \cdot \pi(\omega,a)(s')$.

Observe that the scheduler does not choose an action but a *distribution* over actions. It is well-known [31] that such randomisation brings more power in the context of bisimulations. To the contrary, nature is not allowed to randomise over the set of feasible distributions $\mathscr{E}^{s,a}$. This is in fact not necessary, since the set $\mathscr{E}^{s,a}$ is closed under convex combinations. Finally, a scheduler σ is said to be *deterministic* if $\sigma(\omega)(a) = 1$ for some action a for all finite paths ω.

2.2 Probabilistic Computation Tree Logic (PCTL)

There are various ways how to describe properties of interval *MDP*s. Here we focus on *probabilistic CTL* (PCTL) [15]. The syntax of PCTL state formulas φ and PCTL path formulas ψ is given by:

$$\varphi := true \mid x \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{P}_{\bowtie p}(\psi)$$
$$\psi := \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2 \mid \varphi_1 \mathsf{U}^{\leq k} \varphi_2$$

where $x \in AP$, $p \in [0,1]$ is a rational constant, $\bowtie \in \{\leq, <, \geq, >\}$, and $k \in \mathbb{N}$.

The satisfaction relation for PCTL formulae depends on the way how non-determinism is resolved for the probabilistic operator $\mathsf{P}_{\bowtie p}(\psi)$. When quantifying both the non-determinisms universally, we define the satisfaction relation $s \models_{(\forall)} \varphi$ as follows: $s \models_{(\forall)} x$ if $x \in L(s)$; $s \models_{(\forall)} \neg\varphi$ if not $s \models_{(\forall)} \varphi$; $s \models_{(\forall)} \varphi_1 \wedge \varphi_2$ if both $s \models_{(\forall)} \varphi_1$ and $s \models_{(\forall)} \varphi_2$; and

$$s \models_{(\forall)} \mathsf{P}_{\bowtie p}(\psi) \quad \text{if} \quad \forall\sigma \in \Sigma\ \forall\pi \in \Pi : \quad \Pr_s^{\sigma,\pi}\left[\models_{(\forall)} \psi\right] \bowtie p. \tag{$\forall$}$$

where $\models_{(\forall)} \psi$ denotes the set of infinite paths $\{\omega \in Paths_{inf} \mid \omega \models_{(\forall)} \psi\}$ and the satisfaction relation

[1] Here, $\mathscr{B}$ is the standard σ-algebra over $Paths_{inf}$ generated from the set of all cylinder sets $\{Paths_\omega \mid \omega \in Paths_{fin}\}$. The unique probability measure is obtained by the application of the by extension theorem (see, e.g. [3]).

$\omega \models_{(\forall)} \psi$ for an infinite path $\omega = s_1 s_2 \cdots$ and a path formula ψ is given by:

$$\begin{aligned}
&\omega \models_{(\forall)} \mathsf{X}\varphi && \text{if} \quad s_2 \models \varphi;\\
&\omega \models_{(\forall)} \varphi_1 \mathsf{U}^{\leq k} \varphi_2 && \text{if} \quad \text{there exists } i \leq k \text{ such that } s_i \models_{(\forall)} \varphi_2,\\
& && \qquad \text{and } s_j \models_{(\forall)} \varphi_1 \text{ for every } 0 \leq j < i;\\
&\omega \models_{(\forall)} \varphi_1 \mathsf{U} \varphi_2 && \text{if} \quad \text{there exists } k \in \mathbb{N} \text{ such that } \omega \models_{(\forall)} \varphi_1 \mathsf{U}^{\leq k} \varphi_2.
\end{aligned}$$

It is easy to show that the set $\models_{(\forall)} \psi$ is measurable for any path formula ψ, hence the definition is correct. We explain how the semantics differs for different resolution of non-determinism in the next section.

3 Probabilistic Bimulations for Interval Markov decision processes

Let us fix an interval MDP (S, A, AP, L, I). In this section, we define probabilistic bisimulations for different interpretations of Interval MDP. Namely the bisimulation $\sim_{(\forall)}$ for the cooperative setting and bisimulations $\sim_{(\exists\sigma\forall)}$ and $\sim_{(\exists\pi\forall)}$ for two different applications for the competitive setting. We then show that $\sim_{(\exists\sigma\forall)}$ and $\sim_{(\exists\pi\forall)}$ actually coincide.

3.1 Cooperative resolution of non-determinism

In the context of verification of parallel systems with uncertain transition probabilities, it makes sense to assume that *Scheduler* and *Nature* are resolved *cooperatively* in the most *adversarial* way. This setting yields a bisimulation quite similar to standard probabilistic bisimulation for models with one type of non-determinism [23]. First, let us denote by $s \longrightarrow \mu$ that a transition from s according to μ can be taken cooperatively, i.e. that there is a decision $\rho \in \mathit{Dist}(A)$ of *Scheduler* and decisions $s \xrightarrow{a} \mu_a$ of *Nature* for each a such that $\mu = \sum_{a \in A} \rho(a) \cdot \mu_a$. In other words, $s \longrightarrow \mu$ if $\mu \in \operatorname{conv}\{\mathscr{E}^{s,a} \mid a \in A\}$ where $\operatorname{conv} X$ denotes the convex hull of X.

Definition 3. *Let $R \subseteq S \times S$ be a equivalence relation. We say that R is* probabilistic $(\forall)$-bisimulation *if for any $(s,t) \in R$ we have that $L(s) = L(t)$ and*

$$\begin{aligned}
&\textit{for each} \quad s \longrightarrow \mu\\
&\textit{there is} \quad t \longrightarrow \nu \quad \textit{such that} \quad \mu(\mathscr{C}) = \nu(\mathscr{C}) \textit{ for each equivalence class } \mathscr{C} \in S/R.
\end{aligned}$$

Furthermore, we write $s \sim_{(\forall)} t$ if there is a probabilistic $(\forall)$-bisimulation R such that $(s,t) \in R$.

Intuitively, each (cooperative) step of *Scheduler* and *Nature* from state s needs to be matched by a (cooperative) step of *Scheduler* and *Nature* from state t; symmetrically, s also needs to match t. As a first result, we show that the bisimulation $\sim_{(\forall)}$ preserves the (cooperative) universally quantified PCTL satisfaction $\models_{(\forall)}$.

Theorem 1. *For states $s \sim_{(\forall)} t$ and any PCTL formula φ, we have $s \models_{(\forall)} \varphi$ if and only if $t \models_{(\forall)} \varphi$.*

Dually, the non-determinism could also be resolved *existentially*. This corresponds to the setting where we want to synthesise both the scheduler σ that controls the system and choice of feasible probability distributions π such that σ and π together guarantee a specified behaviour φ. This setting is formalised by the satisfaction relation $\models_{\exists}$ which is defined like $\models_{(\forall)}$ except for the operator $\mathsf{P}_{\bowtie p}(\psi)$ where we set

$$s \models_{(\exists)} \mathsf{P}_{\bowtie p}(\psi) \quad \text{if} \quad \exists \sigma \in \Sigma \ \exists \pi \in \Pi : \quad \Pr_s^{\sigma,\pi}\left[\models_{(\exists)} \psi\right] \bowtie p. \tag{$\exists$}$$

Note that for any formula of the form $\mathsf{P}_{<p}(\psi)$, we have $s \models_\exists \mathsf{P}_{<p}(\psi)$ if and only if we have $s \models_{(\forall)} \neg\mathsf{P}_{\geq p}(\psi)$. This can be easily generalised: for each state formula φ we obtain a state formula $\overline{\varphi}$ such that $s \models_\exists \varphi$ if and only if $s \models_{(\forall)} \overline{\varphi}$ for each state s. Hence $\sim_{(\forall)}$ also preserves $\models_\exists$.

Corollary 1. *For states $s \sim_{(\forall)} t$ and any PCTL formula φ, we have $s \models_{(\exists)} \varphi$ if and only if $t \models_\exists \varphi$.*

3.2 Competitive resolution of non-determinism

As already argued for in Section 1, there are applications where it is natural to interpret the two sources of non-determinism in a competitive way.

Control synthesis under uncertainty In this setting we search for a scheduler σ such that for any nature π, a fixed property φ is satisfied. This corresponds to the satisfaction relation $\models_{(\exists\sigma\forall)}$, obtained similarly from $\models_{(\forall)}$ by replacing the rule $(\forall)$ with

$$s \models_{(\exists\sigma\forall)} \mathsf{P}_{\bowtie p}(\psi) \quad \text{if} \quad \exists\sigma\in\Sigma\ \forall\pi\in\Pi: \quad \Pr_s^{\sigma,\pi}\left[\models_{(\exists\sigma\forall)} \psi\right] \bowtie p. \tag{$\exists\sigma\forall$}$$

As regards bisimulation, the competitive setting is not a common one. We define a bisimulation similar to the alternating bisimulation of [1] applied to non-stochastic two-player games. For a decision $\rho \in \Delta(A)$ of *Scheduler*, let us denote by $s \xrightarrow{\rho} \mu$ that μ is a possible successor distribution, i.e. there are decisions μ_a of *Nature* for each a such that $\mu = \sum_{a\in A} \rho(a)\cdot\mu_a$.

Definition 4. *Let $R \subseteq S\times S$ be an equivalence relation. We say that R is* probabilistic $(\exists\sigma\forall)$-bisimulation *if for any $(s,t)\in R$ we have that $L(s)=L(t)$ and*

$$\begin{aligned}
&\textit{for each} && \rho_s \in \Delta(A)\\
&\textit{there is} && \rho_t \in \Delta(A)\\
&\textit{such that for each} && t \xrightarrow{\rho_t} \nu\\
&\textit{there is} && s \xrightarrow{\rho_s} \mu \quad \textit{such that } \mu(\mathscr{C}) = \nu(\mathscr{C}) \textit{ for each equivalence class } \mathscr{C}\in S/R.
\end{aligned}$$

Furthermore, we write $s \sim_{(\exists\sigma\forall)} t$ if there is a probabilistic $(\exists\sigma\forall)$-bisimulation R such that $(s,t)\in R$.

The exact alternation of quantifiers might be counter-intuitive at first sight. Note that it exactly corresponds to the situation in non-stochastic games [1] and that this bisimulation preserves the PCTL logic with $\models_{(\exists\sigma\forall)}$.

Theorem 2. *For states $s \sim_{(\exists\sigma\forall)} t$ and any PCTL formula φ, we have $s \models_{(\exists\sigma\forall)} \varphi$ if and only if $t \models_{(\exists\sigma\forall)} \varphi$.*

Similarly to Corollary 1, we could define a satisfaction relation with the alternation $\forall\sigma\in\Sigma\ \exists\pi\in\Pi$ that is then preserved by the same bisimulation $\sim_{(\exists\sigma\forall)}$. However, we see no natural application thereof.

Parameter synthesis in parallel systems In this setting, we search for a resolution π of the underspecified probabilities such that for any scheduler σ resolving the interleaving non-determinism, a fixed property φ is satisfied. This corresponds to the satisfaction relation $\models_{(\exists\pi\forall)}$, obtained similarly from $\models_{(\forall)}$ by replacing the rule $(\forall)$ with

$$s \models_{(\exists\pi\forall)} \mathsf{P}_{\bowtie p}(\psi) \quad \text{if} \quad \exists\pi\in\Pi\ \forall\sigma\in\Sigma: \quad \Pr_s^{\sigma,\pi}\left[\models_{(\exists\pi\forall)} \psi\right] \bowtie p. \tag{$\exists\pi\forall$}$$

This yields a definition of bisimulation similar to Definition 4. For a choice $(\mu_a)_{a\in A}$ of underspecified probabilities, let us denote by $s \xrightarrow{(\mu_a)} \mu$ that μ is a possible successor distribution, i.e. there is a decision ρ of *Scheduler* such that $\mu = \sum_{a\in A}\rho(a)\cdot\mu_a$.

Definition 5. *Let $R \subseteq S \times S$ be a symmetric relation. We say that R is* probabilistic $(\exists\pi\forall)$-bisimulation *if for any $(s,t) \in R$ we have that $L(s) = L(t)$ and*

$$\begin{aligned} \textit{for each} \quad & (\mu_a)_{a\in A} \quad \textit{where } s \xrightarrow{a} \mu_a \ \textit{for each } a \in A \\ \textit{there is} \quad & (\nu_a)_{a\in A} \quad \textit{where } t \xrightarrow{a} \nu_a \ \textit{for each } a \in A \\ \textit{such that for each} \quad & t \xrightarrow{(\nu_a)} \nu \\ \textit{there is} \quad & s \xrightarrow{(\mu_a)} \mu \quad \textit{such that} \quad \mu(\mathscr{C}) = \nu(\mathscr{C}) \textit{ for each equivalence class } \mathscr{C} \in S/R, \end{aligned}$$

Furthermore, we write $s \sim_{(\exists\pi\forall)} t$ if there is a probabilistic $(\exists\pi\forall)$-bisimulation R such that $(s,t) \in R$.

The fact that this bisimulation preserves $\models_{(\exists\pi\forall)}$ can be proven analogously to Theorem 2.

Theorem 3. *For states $s \sim_{(\exists\pi\forall)} t$ and any PCTL formula φ, we have $s \models_{(\exists\pi\forall)} \varphi$ if and only if $t \models_{(\exists\pi\forall)} \varphi$.*

As a final result of this section, we show that these two bisimulations coincide.

Theorem 4. *We have $\sim_{(\exists\sigma\forall)} \;=\; \sim_{(\exists\pi\forall)}$.*

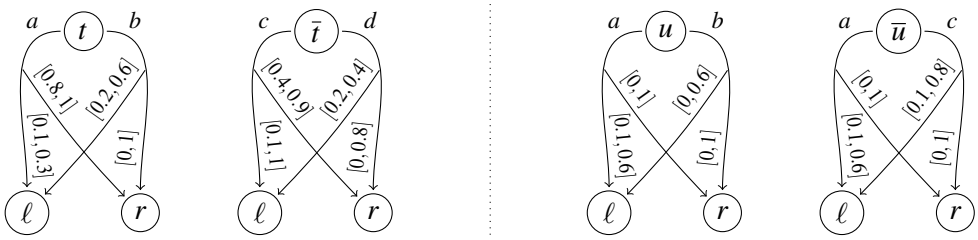
Thanks to this result, we denote from now on these coinciding bisimulations by $\sim_{(\exists\forall)}$. As a concluding remark, note that Definitions 3, 4 and 5 can be seen as the conservative extension of probabilistic bisimulation for (state-labelled) MDPs. To see that assume the set of uncertainty for every transition is a singleton. Since there is only one choice for the nature, the role of nature can be safely removed from the definitions. Moreover, it is worthwhile to note that Theorems 1, 2 and 3 show the soundness of the probabilistic bisimulation definitions with respect to PCTL. Unfortunately, it is shown in [31, 32] that probabilistic bisimulation for probabilistic automata is finer than PCTL equivalence which leads to the incompleteness in general. Since MDPs can be seen as a subclass of PAs, it is not difficult to see that the incompleteness holds also for MDPs.

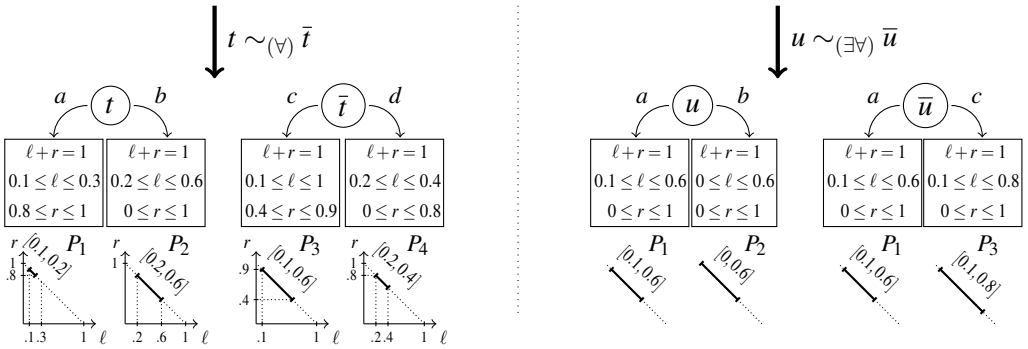
We also remark that the notions $\sim_{(\forall)}$ and $\sim_{(\exists\forall)}$ are incomparable, as it is for instance observable in Example 1.1. It is shown in the example that $t \sim_{(\forall)} \bar{t}$ and $u \sim_{(\exists\forall)} \bar{u}$. However it is not hard to verify that $t \not\sim_{(\exists\forall)} \bar{t}$ and $u \not\sim_{(\forall)} \bar{u}$. For the latter, notice that for example u can evolve to r with probability one by taking action b, whereas $\bar{u}$ cannot simulate. The former is noticeable in the situation where the controller wants to maximise the probability to reach r, but the nature declines. In this case t chooses action b and the nature let it go to r with probability 0.8. Nevertheless the nature can prevent $\bar{t}$ to evolve into r with probability more than 0.6, despite the fact which action has been chosen by $\bar{t}$.

4 Algorithms

In this section, we give algorithms for computing bisimulations $\sim_{(\forall)}$ and $\sim_{(\exists\forall)}$. We show that computing bisimulations in both cases is fixed-parameter tractable.

Example 4.1. Let us start by illustrating the ideas on Example 1.1 from Section 1.





The general sketch of the algorithm is as follows. We need to construct the polytopes of probability distributions offered by the actions; in our examples the polytopes are just line segments in two-dimensional space. We get $t \sim_{(\forall)} \bar{t}$ since the *convex hull* of P_1 and P_2 equals to the *convex hull* of P_3 and P_4. Similarly, we get $u \sim_{(\exists\forall)} \bar{u}$ since u and $\bar{u}$ have the same set of *minimal* polytopes w.r.t. set inclusion.

Let us state the results formally. Let us fix $\mathfrak{M} = (S, A, AP, L, I)$ where b is the maximal number of different actions $\max_{s\in S} |\{I(s,a,\cdot) \mid a \in A\}|$, f is the maximal support of an action $\max_{s\in S, a\in A} |\{s' \mid I(s,a,s') \neq [0,0]\}|$, and $|\mathfrak{M}|$ denotes the size of the representation using adjacency lists for non-zero elements of I where we assume that the interval bounds are rational numbers encoded symbolically in binary.

Theorem 5. *There is an algorithm that computes $\sim_{(\forall)}$ in time polynomial in $|\mathfrak{M}|$ and exponential in f. There is also an algorithm that computes $\sim_{(\exists\forall)}$ in time polynomial in $|\mathfrak{M}|$ and exponential in f and b.*

Computing both bisimulations follows the standard partition refinement approach [4, 19, 26], formalized by the procedure `Bisimulation` in Algorithm 1. Namely, we start with R being the complete relation and iteratively remove from R pairs of states that violate the definition of bisimulation with respect to R. The core part is finding out whether two states "violate the definition of bisimulation". This is where the algorithms for the two bisimulations differ.

4.1 Cooperative resolution of non-determinism of the bisimulation $\sim_{(\forall)}$

Let us first address $\sim_{(\forall)}$ where the violation is checked by the procedure $\texttt{Violate}_{(\forall)}$. We show that this amounts to checking inclusion of polytopes defined as follows. Recall that for $s \in S$ and an action $a \in A$, $\mathscr{E}^{s,a}$ denotes the polytope of feasible successor distributions over *states* with respect to taking the action a in the state s. By $\mathscr{E}^{s,a}_R$, we denote the polytope of feasible successor distributions over *equivalence classes* of R with respect to taking the action a in the state s. Formally, for $\mu \in \Delta(S/R)$ we set $\mu \in \mathscr{E}^{s,a}_R$ if we have

$$\mu(\mathscr{C}) \in \Big[\sum_{s'\in\mathscr{C}} \inf I(s,a,s') \sum_{s'\in\mathscr{C}} \sup I(s,a,s')\Big] \qquad \text{for each } \mathscr{C} \in S/R.$$

Note that we require that the probability of each class $\mathscr{C}$ must be in the interval of the sum of probabilities that can be assigned to states of $\mathscr{C}$. Furthermore, we define $\mathscr{E}^s_R$ as the convex hull of $\bigcup_{a\in A} \mathscr{E}^{s,a}_R$. It is the set of feasible successor distributions over S/R with respect to taking an *arbitrary* distribution over actions in state s. As specified in the procedure $\texttt{Violate}_{(\forall)}$, we show that it suffices to check equality of these polytopes.

Proposition 1. *We have $s \sim_{(\forall)} t$ if and only if $L(s) = L(t)$ and $\mathscr{E}^s_{\sim_{(\forall)}} = \mathscr{E}^t_{\sim_{(\forall)}}$.*

Bisimulation($\mathfrak{M}$)

$R \leftarrow \{(s,t) \in S \times S \mid L(s) = L(t)\}$;
repeat
  $R' \leftarrow R$;
  for all $s \in S$ **do**
    $D \leftarrow \emptyset$;
    for all $t \in [s]_R$ **do**
      if Violate(s,t,R)
        $D \leftarrow D \cup \{t\}$;
    split $[s]_R$ in R into D and $[s]_R \setminus D$;
until $R = R'$;
return R;

Violate$_{(\forall)}(s,t,R)$

return $\mathscr{E}_R^s \neq \mathscr{E}_R^t$;

Violate$_{(\exists\forall)}(s,t,R)$

$S, T \leftarrow \emptyset$;
for all $a \in A$ **do**
  if Minimal(s,a,R) // $\mathscr{E}_R^{s,a}$ strictly m.?
    $S \leftarrow S \cup \{\mathscr{E}_R^{s,a}\}$;
  if Minimal(t,a,R)
    $T \leftarrow T \cup \{\mathscr{E}_R^{t,a}\}$;
return $S \neq T$;

Minimal(s,a,R)

$k \leftarrow |A| - 1$;
$C_1, \ldots, C_k \leftarrow$ compute the sets of corners of other polytopes;
$B \leftarrow (\mathbf{1}, -\mathbf{1})$; // constraints on ρ such that $B\rho + d \geq 0$ implies $\mathscr{E}_R^{s,\rho} \subseteq \mathscr{E}_R^{s,a}$
$d \leftarrow (-1, 1)$; // initially, $B\rho + d \geq 0$ implies $\sum \rho = 1$, i.e. $\rho \in \Delta(\{1, \ldots, k\})$
for all $c_1, \ldots, c_k \in C_1 \times \cdots \times C_k$ **do**
  $R \leftarrow \emptyset$;
  for all intersections $\mathbf{x}$ of $\mathscr{E}_R^{s,a}$ with the line segment from v_i to v_j for some $i \neq j$ **do**
    $R \leftarrow R \cup \{(r_1, \cdots, r_k)\}$ where $r_i \cdot c_i + r_j \cdot c_j = \mathbf{x}$ and $r_\ell = 0$ for $\ell \notin \{i, j\}$;
  for all facets F of the convex hull of R **do**
    add to matrices B, d a constraint corresponding to the half-space given by F that includes R;
return ($B\rho + d = 0$ not feasible); // no intersection of the convex hulls of all sets R?

Algorithm 1: Probabilistic bisimulation algorithm for interval *MDP*s

Proof. Let us first introduce one notation. For each distribution $\mu \in \Delta(S)$, let $\overline{\mu} \in \Delta(S/\sim_{(\forall)})$ denote the corresponding distribution such that $\overline{\mu}(\mathscr{C}) = \sum_{s \in \mathscr{C}} \mu(s)$. As regards the "if" part, for each choice $s \longrightarrow \mu$, we have $\overline{\mu} \in \mathscr{E}^s_{\sim_{(\forall)}}$. Similarly, for each $\rho \in \mathscr{E}^t_{\sim_{(\forall)}}$, there is a choice $t \longrightarrow \nu$ such that $\overline{\nu} = \rho$. Hence, $s \sim_{(\forall)} t$. As regards the "only if" part, let us assume that there is a distribution ρ over equivalence classes such that, say $\rho \in \mathscr{E}^s_{\sim_{(\forall)}} \setminus \mathscr{E}^t_{\sim_{(\forall)}}$. There must be a choice $s \longrightarrow \mu$ such that $\overline{\mu} = \rho$ and there is no choice $t \longrightarrow \nu$ such that $\overline{\nu} = \rho$. Hence, $s \not\sim_{(\forall)} t$. □

Complexity Given an *IMDP* $\mathfrak{M}$, let $|S| = n$, $|A| = m$, b be the maximal number of different actions $\max_{s \in S} |\{I(s,a,\cdot) \mid a \in A\}|$, and f be the maximal support of an action $\max_{s \in S, a \in A} |\{s' \mid I(s,a,s') \neq [0,0]\}|$.

It is easy to see that the procedure Violate$_{(\forall)}$ is called at most n^3-times. Each polytope $\mathscr{E}_R^{s,a}$ has at most $C = f \cdot 2^{f-1}$ corners, computing the convex hull $\mathscr{E}_R^s$ takes $\mathscr{O}((bC)^2)$ time [5]. Checking inclusion of two polytopes then can be done in time polynomial [34] in the number of corners of these two polytopes. In total, computing of $\sim_{(\forall)}$ can be done in time $|\mathfrak{M}|^{\mathscr{O}(1)} \cdot 2^{\mathscr{O}(f)}$.

4.2 Competitive resolution of non-determinism of the bisimulation $\sim_{(\exists\forall)}$

In this case, the violation of bisimilarity of s and t with respect to R is addressed by the procedure Violate$_{(\exists\forall)}$. Here, we check that s and t have the same set of strictly minimal polytopes. For a state s, an action $a \in A$, and an equivalence $R \subseteq S \times S$, we say that $\mathscr{E}_R^{s,a}$ is *strictly minimal* if no convex combination of the remaining polytopes of s is a subset of $\mathscr{E}_R^{s,a}$. More precisely, if for no distribution $\rho \in \Delta(A \setminus \{a\})$,

we have $\mathscr{E}_R^{s,\rho} \subseteq \mathscr{E}_R^{s,a}$ where $\mathscr{E}_R^{s,\rho}$ denotes the polytope $\{\sum_{b\in A\setminus\{a\}} \rho(b)\cdot \mathbf{x}_b \mid \text{each } \mathbf{x}_b \in \mathscr{E}_R^{s,b}\}$.

Proposition 2. *We have $s \sim_{(\exists\forall)} t$ if and only if $L(s)=L(t)$ and $\{\mathscr{E}^{s,a}_{\sim_{(\exists\forall)}} \mid a\in A, \mathscr{E}^{s,a}_{\sim_{(\exists\forall)}}$ is strictly minimal$\} = \{\mathscr{E}^{t,a}_{\sim_{(\exists\forall)}} \mid a\in A, \mathscr{E}^{t,a}_{\sim_{(\exists\forall)}}$ is strictly minimal$\}$.*

Proof. We first address the "if" part. For each choice of *Nature* $(\mu_a)_{a\in A}$ where each $s \xrightarrow{a} \mu_a$, let $M = \{\overline{\mu}_a \mid a\in A\}$ and $M' \subseteq M$ be the subset where each distribution lies within some strictly minimal polytope $\mathscr{E}^{s,b}_{\sim_{(\exists\forall)}}$. Because the strictly minimal polytopes coincide, we can construct a choice of *Nature* $N=(\nu_a)_{a\in A}$ such that $N=\{\overline{\nu}_a \mid a\in A\} = M'$. Because $N\subseteq M$, it is easy to see that for each $t \xrightarrow{(\nu_a)} \nu$ there is $s \xrightarrow{(\mu_a)} \mu$ such that $\mu(\mathscr{C}) = \nu(\mathscr{C})$ for each $\mathscr{C}\in S/R$.

As regards the "only if" part, let us assume that there is, say in t, a strictly minimal polytope $\mathscr{E}^{t,b}_{\sim_{(\exists\forall)}}$ that is not in the set of strictly minimal polytopes for s. There is a choice of *Nature* $(\mu_a)_{a\in A}$ for state s such that no convex combination of elements of $M=\{\overline{\mu}_a \mid a\in A\}$ lies in $\mathscr{E}^{t,b}_{\sim_{(\exists\forall)}}$; in particular no element of M lies in $\mathscr{E}^{t,b}_{\sim_{(\exists\forall)}}$. For any choice of *Nature* $(\nu_a)_{a\in A}$ for state t, $\overline{\nu}_b$ is not a convex combination of elements from M. Thus, if *Scheduler* chooses action b, there is no $s \xrightarrow{(\mu_a)} \mu$ such that $\mu(\mathscr{C}) = \nu_b(\mathscr{C})$ for each $\mathscr{C}\in S/R$ and it does *not* hold $s\sim_{(\exists\forall)} t$. □

Next, we need to address how to compute whether a polytope is strictly minimal. We construct B and d such that $B\rho + d \geq 0$ implies $\mathscr{E}_R^{s,\rho} \subseteq \mathscr{E}_R^{s,a}$. Checking of strictly minimality then reduces to checking feasibility of this linear system. The system gets constructed iteratively. Let $P_1,\cdots,P_k$ denote the polytopes corresponding to all actions in s except for a. We enumerate all combinations $(c_1,\ldots,c_k)\in C(P_1)\times\cdots\times C(P_k)$ of corners of the polytopes. For each such combination we add into B and d new constraints $B_{(c_1,\ldots,c_k)}$ and $d_{(c_1,\ldots,c_k)}$ such that for any ρ satisfying $B_{(c_1,\ldots,c_k)}\rho + d_{(c_1,\ldots,c_k)} \geq 0$ we have $\sum \rho_i c_i \in \mathscr{E}_R^{s,a}$. For details, see the procedure `Minimal` in Algorithm 1.

Proposition 3. *We have $B\rho+d\geq 0$ is not feasible if and only if $\mathscr{E}_R^{s,a}$ is strictly minimal where the rows of B and d are obtained as a union of rows*

$$\begin{aligned} B &= \{\mathbf{1},-\mathbf{1}\} \cup \bigcup \{B_{(c_1,\ldots,c_k)} \mid (c_1,\ldots,c_k) \in C(P_1)\times\cdots\times C(P_k)\} \\ d &= \{-1,1\} \cup \bigcup \{d_{(c_1,\ldots,c_k)} \mid (c_1,\ldots,c_k) \in C(P_1)\times\cdots\times C(P_k)\}. \end{aligned}$$

Proof. Let ρ be any feasible solution of the system. It is easy to see that $\mathscr{E}_R^{s,\rho} \subseteq \mathscr{E}_R^{s,a}$ since $\mathscr{E}_R^{s,\rho}$ is convex and since all corners of $\mathscr{E}_R^{s,\rho}$ (obtained as a convex ρ-combination of corners of all $\mathscr{E}_R^{s,b}$) lie within $\mathscr{E}_R^{s,a}$. Hence, $\mathscr{E}_R^{s,a}$ is not strictly minimal. As regards the other direction, let $\mathscr{E}_R^{s,a}$ be not strictly minimal. By definition, there is a distribution ρ over the remaining actions in s such that $\mathscr{E}_R^{s,\rho} \subseteq \mathscr{E}_R^{s,a}$. Then, this distribution ρ must satisfy $B\rho+d\geq 0$. □

Complexity Again let $|S|=n$, $|A|=m$, b be the maximal number of different actions $\max_{s\in S} |\{I(s,a,\cdot) \mid a\in A\}|$, and f be the maximal support of an action $\max_{s\in S,a\in A} |\{s' \mid I(s,a,s') \neq [0,0]\}|$.

Again, $\texttt{Violate}_{(\exists\forall)}$ is called at most n^3 times. The procedure $\texttt{Violate}_{(\exists\forall)}$ is then linear in m and in the complexity of `Minimal`. There are at most $(f\cdot 2^{f-1})^b$ combinations of corners of the polytopes. For each such combination, $b(b-1)$ times the intersection points of a line and a polytope are computed (in time polynomial in $|\mathfrak{M}|$), and at most $f!$ facets of the resulting polytope R are inspected. Overall, computing of $\sim_{(\exists\forall)}$ can be done in time $|\mathfrak{M}|^{\mathscr{O}(1)}\cdot 2^{\mathscr{O}(f^2 b)}$.

5 Case Study

As a case study, we consider a model of Carrier Sense Multiple Access with Collision Detection (CSMA / CD), which is an access control on a shared medium, used mostly in early Ethernet technology. In this scheme multiple devices can be connected to a shared bus. Multiple attempts at the same time to grab bus access leads to collision. At this point, the senders in collision probabilistically schedule a retransmission according to exponential back-off algorithm. The algorithm uniformly determines a delay before the next retransmission, which is between 0 to $2^n - 1$ time slots after occurrence of n-th collision. After a pre-specified number of failed retransmissions, a sender aborts the sending process.

There are two sources of uncertainty in the model. Uncertainty in sending data lies in the fact that the exact probability of sending a message from a sender could be unknown. Instead it is within an interval. The other source comes from imprecise information about collision. If two nodes try to send a frame at the slightly same time, a collision will happen. Conversely it will not happen, when the later transmitter checks the bus and detects it occupied. Since the exact probability of a collision occurrence depends on many parameters and is likely unknown, it is expressed as an uncertain interval rather than an exact value in the model.

Concurrent execution of the node and the bus processes assembles the CSMA/CD model. To this end we need a formalism that supports communication among components via parallel composition. We thus consider a subclass of abstract PAs [9] with interval constraints on probabilities. The subclass in general is not closed under parallel composition. The problem arises when two actions exhibiting uncertainty want to synchronise. Parallel composition in this case imposes some interdependency between the choices of the composed action, which cannot be expressed by a simple interval bound and needs to be expressed by more complicated polynomial constraints. Nevertheless by excluding synchronisation of actions containing uncertainty, abstract PAs with interval constraints feature closure under parallel composition and thereby allow compositional modelling. This is of course not a strict restriction, because we can always shift uncertainty to the actions that are not subject to parallel composition by introducing proper auxiliary states and transitions. In our case study all components are in this subclass and respects the restriction, as uncertainty prevails on actions that are not subject to parallel composition. Consequently it enables us to utilise compositional system design by using existing tools. Since the model arising from parallel composition is not subject to any further communication, we can close it and obtain an IMDP at the end.

We use process algebra prCRL [21], implemented in tool scoop [35], for compositional modelling of CSMA/CD. The model has two parameters: number of nodes attached to the bus and maximum collision allowed before abortion. As we are interested in model checking of a model arisen from parallel composition we apply the semantics of bisimulation in cooperative way, namely $\sim_{(\forall)}$. The state space is generated by scoop and then the bisimulation quotient is computed. Since the maximum size f of the set supported by uncertain transitions is two, the algorithm of Section 4 is tractable. Reduction in state and transition space gained after bisimulation minimisation is reported in Table 1. As shown in the table, the reduction of both state and transition space increases when putting more nodes in the network. Indeed, then there are more nodes performing similar activities and thereby increasing the symmetry in the model. On the other hand, increasing the maximum number of collisions allows the nodes to more likely send frames at different time slots. As a result it decreases the symmetry and then the reduction factor.

Table 1: Impact of bisimulation minimisation on CSMA/CD model[2]

Node #	Max collision #	Original Model		Minimised Model		Reduction Factor	
		State #	Transition #	States #	Transition #	For states	For transitions
2	1	233	466	120	220	48%	53%
	2	497	988	310	581	38%	41%
	3	865	1858	576	1186	33%	36%
3	1	4337	10074	1244	2719	71%	73%
	2	52528	125715	18650	42795	64%	66%
	3	239213	619152	90492	225709	62%	64%
4	1	60357	154088	10904	27308	82%	82%
	2	1832005	4876636	421570	1112129	77%	77%
5	1	751905	2043090	90538	248119	88%	88%

6 Conclusion

In this paper, we study strong bisimulations for interval MDPs. In these models there are two sources of non-determinism and we deal with different interpretations of these non-determinisms. This yields two different bisimulations and we give decision algorithms for both of them.

Note that our decision algorithms can be easily adapted to the slightly broader setting of uncertain MDPs with rectangular uncertainty sets [25]. In this setting, a general convex polytope (not necessarily induced by intervals) is associated to each action in each state. Still, it is assumed that transition probabilities from different states or under different actions are independent.

First open question for future work is the exact complexity of our decision problems. One way to address this question is to prove NP-hardness of the general problem. Another way is to identify interesting subclasses of interval MDPs for that a polynomial-time algorithm exists. Second direction for future work is to address a richer formalism for uncertainties (such as polynomial constraints or even parameters appearing in multiple states/actions). Third, compositional modelling over interval models also deserves a more systematic treatment. Understanding better the ways how large models with interval uncertainties can be composed, may bring further ideas for efficient analysis of these models.

Acknowledgements We would like to thank Holger Hermanns for inspiring discussions. This work has been supported by the DFG as part of the SFB/TR 14 "Automatic Verification and Analysis of Complex Systems" (AVACS), by the Czech Science Foundation under the grant agreement no. P202/12/G061, by the DFG/NWO Bilateral Research Programme ROCKS, and by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS) and 318490 (SENSATION).

References

[1] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman & Moshe Y. Vardi (1998): *Alternating Refinement Relations*. In: *CONCUR*, *LNCS* 1466, Springer, pp. 163–178. Available at `http://dx.doi.org/10.1007/BFb0055622`.

[2] Michael Benedikt, Rastislav Lenhardt & James Worrell (2013): *LTL Model Checking of Interval Markov Chains*. In: *TACAS*, *LNCS* 7795, Springer, pp. 32–46. Available at `http://dx.doi.org/10.1007/978-3-642-36742-7_3`.

[2] The computation time does not reflect the complexity of the algorithm, as it is greatly effected by the file exchange between the tools used for modelling and bisimulation minimisation. Hence it is omitted from the table.

[3] Patrick Billingsley (1979): *Probability and Measure*. John Wiley and Sons, New York, Toronto, London.

[4] Stefano Cattani & Roberto Segala (2002): *Decision Algorithms for Probabilistic Bisimulation*. In: *CONCUR*, *LNCS* 2421, pp. 371–385. Available at http://dx.doi.org/10.1007/3-540-45694-5_25.

[5] Donald R. Chand & Sham S. Kapur (1970): *An Algorithm for Convex Polytopes*. *J. ACM* 17(1), pp. 78–86. Available at http://dx.doi.org/10.1145/321556.321564.

[6] Krishnendu Chatterjee, Siddhesh Chaubal & Pritish Kamath (2012): *Faster Algorithms for Alternating Refinement Relations*. In: *CSL*, *LIPIcs* 16, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 167–182. Available at http://dx.doi.org/10.4230/LIPIcs.CSL.2012.167.

[7] Krishnendu Chatterjee, Koushik Sen & Thomas A. Henzinger (2008): *Model-Checking omega-Regular Properties of Interval Markov Chains*. In: *FoSSaCS*, *LNCS* 4962, Springer, pp. 302–317. Available at http://dx.doi.org/10.1007/978-3-540-78499-9_22.

[8] Taolue Chen, Tingting Han & Marta Z. Kwiatkowska (2013): *On the complexity of model checking interval-valued discrete time Markov chains*. *Inf. Process. Lett.* 113(7), pp. 210–216. Available at http://dx.doi.org/10.1016/j.ipl.2013.01.004.

[9] Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher & Andrzej Wasowski (2011): *Abstract Probabilistic Automata*. In: *VMCAI*, *LNCS* 6538, Springer, pp. 324–339. Available at http://dx.doi.org/10.1007/978-3-642-18275-4_23.

[10] Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher & Andrzej Wasowski (2011): *New Results on Abstract Probabilistic Automata*. In: *ACSD*, IEEE, pp. 118–127. Available at http://doi.ieeecomputersociety.org/10.1109/ACSD.2011.10.

[11] Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen & Andrzej Wasowski (2011): *Decision Problems for Interval Markov Chains*. In: *LATA*, *LNCS* 6638, Springer, pp. 274–285. Available at http://dx.doi.org/10.1007/978-3-642-21254-3_21.

[12] Harald Fecher, Martin Leucker & Verena Wolf (2006): *Don't Know in Probabilistic Systems*. In: *SPIN*, *LNCS* 3925, Springer, pp. 71–88. Available at http://dx.doi.org/10.1007/11691617_5.

[13] Robert Givan, Sonia M. Leach & Thomas L. Dean (2000): *Bounded-parameter Markov decision processes*. *Artif. Intell.* 122(1-2), pp. 71–109. Available at http://dx.doi.org/10.1016/S0004-3702(00)00047-3.

[14] Ernst Moritz Hahn, Tingting Han & Lijun Zhang (2011): *Synthesis for PCTL in Parametric Markov Decision Processes*. In: *NASA Formal Methods*, *LNCS* 6617, Springer, pp. 146–161. Available at http://dx.doi.org/10.1007/978-3-642-20398-5_12.

[15] Hans Hansson & Bengt Jonsson (1994): *A Logic for Reasoning about Time and Reliability*. *Formal Asp. Comput.* 6(5), pp. 512–535. Available at http://dx.doi.org/10.1007/BF01211866.

[16] Vahid Hashemi, Hassan Hatefi & Jan Krčál (2014): *Probabilistic Bisimulations for PCTL Model Checking of Interval MDPs (extended version)*. Available at http://arxiv.org/abs/1403.2864.

[17] Garud N. Iyengar (2005): *Robust Dynamic Programming*. *Math. Oper. Res.* 30(2), pp. 257–280. Available at http://dx.doi.org/10.1287/moor.1040.0129.

[18] Bengt Jonsson & Kim Guldstrand Larsen (1991): *Specification and Refinement of Probabilistic Processes*. In: *LICS*, IEEE Computer Society, pp. 266–277. Available at http://dx.doi.org/10.1109/LICS.1991.151651.

[19] Paris C. Kanellakis & Scott A. Smolka (1990): *CCS Expressions, Finite State Processes, and Three Problems of Equivalence*. *Inf. Comput.* 86(1), pp. 43–68. Available at http://dx.doi.org/10.1016/0890-5401(90)90025-D.

[20] Joost-Pieter Katoen, Daniel Klink & Martin R. Neuhäußer (2009): *Compositional Abstraction for Stochastic Systems*. In: *FORMATS*, *LNCS* 5813, Springer, pp. 195–211. Available at http://dx.doi.org/10.1007/978-3-642-04368-0_16.

[21] Joost-Pieter Katoen, Jaco van de Pol, Mariëlle Stoelinga & Mark Timmer (2010): *A Linear Process-Algebraic Format for Probabilistic Systems with Data*. In: *ACSD*, IEEE Computer Society, pp. 213–222. Available at http://doi.ieeecomputersociety.org/10.1109/ACSD.2010.18.

[22] Igor Kozine & Lev V. Utkin (2002): *Interval-Valued Finite Markov Chains*. *Reliable Computing* 8(2), pp. 97–113. Available at http://dx.doi.org/10.1023/A:1014745904458.

[23] Kim Guldstrand Larsen & Arne Skou (1991): *Bisimulation through Probabilistic Testing*. *Inf. Comput.* 94(1), pp. 1–28. Available at http://dx.doi.org/10.1016/0890-5401(91)90030-6.

[24] Robin Milner (1989): *Communication and concurrency*. PHI Series in computer science, Prentice Hall.

[25] Arnab Nilim & Laurent El Ghaoui (2005): *Robust Control of Markov Decision Processes with Uncertain Transition Matrices*. *Operations Research* 53(5), pp. 780–798. Available at http://dx.doi.org/10.1287/opre.1050.0216.

[26] Robert Paige & Robert E. Tarjan (1987): *Three Partition Refinement Algorithms*. *SIAM J. on Computing* 16(6), pp. 973–989. Available at http://dx.doi.org/10.1137/0216062.

[27] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: *FOCS*, IEEE Computer Society, pp. 46–57. Available at http://doi.ieeecomputersociety.org/10.1109/SFCS.1977.32.

[28] Alberto Puggelli, Wenchao Li, Alberto L. Sangiovanni-Vincentelli & Sanjit A. Seshia (2013): *Polynomial-Time Verification of PCTL Properties of MDPs with Convex Uncertainties*. In: *CAV*, *LNCS* 8044, Springer, pp. 527–542. Available at http://dx.doi.org/10.1007/978-3-642-39799-8_35.

[29] Martin L. Puterman (1994): *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st edition. John Wiley & Sons, Inc., New York, NY, USA. Available at http://dx.doi.org/10.1002/9780470316887.

[30] Matthias Ringwald (2009): *Reducing Uncertainty in Wireless Sensor Networks - Network Inspection and Collision-Free Medium Access*. Ph.D. thesis, ETH Zurich, Zurich, Switzerland.

[31] Roberto Segala (1995): *Modeling and Verification of Randomized Distributed Real-Time Systems*. Ph.D. thesis, Laboratory for Computer Science, Massachusetts Institute of Technology.

[32] Roberto Segala & Nancy A. Lynch (1994): *Probabilistic Simulations for Probabilistic Processes*. In: *CONCUR*, *LNCS* 836, pp. 481–496. Available at http://dx.doi.org/10.1007/BFb0015027.

[33] Koushik Sen, Mahesh Viswanathan & Gul Agha (2006): *Model-Checking Markov Chains in the Presence of Uncertainties*. In: *TACAS*, *LNCS* 3920, Springer, pp. 394–410. Available at http://dx.doi.org/10.1007/11691372_26.

[34] K. Subramani (2009): *On the Complexities of Selected Satisfiability and Equivalence Queries over Boolean Formulas and Inclusion Queries over Hulls*. *JAMDS* 2009. Available at http://dx.doi.org/10.1155/2009/845804.

[35] Mark Timmer (2011): *SCOOP: A Tool for SymboliC Optimisations of Probabilistic Processes*. In: *QEST*, IEEE Computer Society, pp. 149–150. Available at http://doi.ieeecomputersociety.org/10.1109/QEST.2011.27.

[36] Eric M. Wolff, Ufuk Topcu & Richard M. Murray (2012): *Robust control of uncertain Markov Decision Processes with temporal logic specifications*. In: *CDC*, IEEE, pp. 3372–3379. Available at http://dx.doi.org/10.1109/CDC.2012.6426174.

[37] Di Wu & Xenofon D. Koutsoukos (2008): *Reachability analysis of uncertain systems using bounded-parameter Markov decision processes*. *Artif. Intell.* 172(8-9), pp. 945–954. Available at http://dx.doi.org/10.1016/j.artint.2007.12.002.

[38] Wang Yi (1994): *Algebraic Reasoning for Real-Time Probabilistic Processes with Uncertain Information*. In: *FTRTFT*, *LNCS* 863, Springer, pp. 680–693. Available at http://dx.doi.org/10.1007/3-540-58468-4_190.

Setting Parameters for Biological Models With ANIMO

Stefano Schivo
Formal Methods and Tools
Faculty of EEMCS
University of Twente
Enschede, The Netherlands
s.schivo@utwente.nl

Jetse Scholma Marcel Karperien Janine N. Post
Developmental BioEngineering
MIRA Institute for Biomedical Technology and Technical Medicine
University of Twente
Enschede, The Netherlands
{j.scholma, h.b.j.karperien, j.n.post}@utwente.nl

Jaco van de Pol Rom Langerak*
Formal Methods and Tools
Faculty of EEMCS
University of Twente
Enschede, The Netherlands
{j.c.vandepol, r.langerak}@utwente.nl

ANIMO (Analysis of Networks with Interactive MOdeling) is a software for modeling biological networks, such as e.g. signaling, metabolic or gene networks. An ANIMO model is essentially the sum of a network topology and a number of interaction parameters. The topology describes the interactions between biological entities in form of a graph, while the parameters determine the speed of occurrence of such interactions.

When a mismatch is observed between the behavior of an ANIMO model and experimental data, we want to update the model so that it explains the new data. In general, the topology of a model can be expanded with new (known or hypothetical) nodes, and enables it to match experimental data. However, the unrestrained addition of new parts to a model causes two problems: models can become too complex too fast, to the point of being intractable, and too many parts marked as "hypothetical" or "not known" make a model unrealistic. Even if changing the topology is normally the easier task, these problems push us to try a better parameter fit as a first step, and resort to modifying the model topology only as a last resource.

In this paper we show the support added in ANIMO to ease the task of expanding the knowledge on biological networks, concentrating in particular on the parameter settings.

1 Introduction

The investigation of biological processes relies on computational support on a daily basis. This happens not only because of the extremely large amount of data generated in the *-omics era*, but also because many processes are simply too complex to be understood without appropriate modeling tools. For this reason, systems biology [25] has become more and more important in the last several years.

A single biological network such as a signaling or gene network, may involve up to hundreds of different molecular species. As it would be very difficult to understand the dynamic behavior of such networks just by looking at their static representations, many tools were built [8, 10, 24, 22, 7, 16] to help the biologists define models of *executable biology* [12]. ANIMO (Analysis of Networks with Interactive MOdeling [26]) is one of such tools. Its primary objective is to let the expert biologists work directly on the formalization of their knowledge, supporting the generation of new insights on the studied processes [27]. An ANIMO model is formed by two main parts: the network topology and

*Corresponding author

É. André and G. Frehse (Eds.): 1st International Workshop
on Synthesis of Continuous Parameters (SynCoP'14).
EPTCS 145, 2014, pp. 35–47, doi:10.4204/EPTCS.145.5

the parameters. The topology describes which biological components are included in the model, and which are the interactions we want to represent. The parameters define the rate of occurrence of such interactions, which are described based on simplified kinetic formulae.

Proteins are normally expressed at different concentrations in different individuals of the same species, and yet the overall behavior of their biological networks does not differ significantly. This phenomenon has led to the notion that biological networks are inherently *robust* [1, 20]. In modeling terms, this means that, if a model is a close representation of a biological network, most of the parameters of that model can vary inside a certain interval without influencing the qualitative behavior of the whole network. The interactive approach of ANIMO is based on the assumption of robustness, as our tool is mainly aimed at the development of network models with a focus on the topology. Ideally, the biologist can "play" with the topology of a network, working towards matching the qualitative behavior of experimental data, rather than precisely reproducing it. However, it is not our intention to concentrate exclusively on the network topology: in many cases a better parameter choice can improve the behavior of a network more than the addition of new components. Indeed, making an unnecessarily complex model could reduce its usefulness both in terms of analysis performances and closeness to reality. The first problem is simply due to the complexity of a network, which would require more and more computational resources to be analysed[1]. Realism of network models is more related to their ultimate usefulness: a model that explains a particular behavior very well but contains many nodes marked as "unknown" has little applicability, as its connection with known processes is very loose. Therefore, it is desirable that a better parameter set is regularly sought for during the design cycle of a complex biological network model. Some support for parameter choice was already provided in the first versions of ANIMO. We present here an extended set of tools aimed at achieving a closer fit between ANIMO models and experimental data. A guideline on how to use these tools to get the best results will also be presented as an ideal workflow. Thanks to the better awareness on parameter choice gained through this new extension of ANIMO, the biologists will be able to judge more easily which are the most promising topologies for a network, and thus drive the experimental research more efficiently.

2 ANIMO models

The starting point of ANIMO is the traditional static representation of biological networks, which can be easily drawn and managed in softwares like Cytoscape [17]. Indeed, ANIMO was implemented as a plug-in to Cytoscape, with the aim of adding dynamics to the static representation of biological networks, and thus allow for analysis on the behavior of such networks. The user interface of ANIMO can be seen in Figure 1a: at the center is the Network panel, where the network model is represented in the familiar nodes-edges form used in the domain of biology. Models in ANIMO are *activity-based*, in the sense that nodes have their activity as main property, and interactions among nodes change the activity of their targets. The concept of activity can be intended for example as a generic post-translational modification a molecule can undergo to change its function. In the case of a kinase, the phosphorylated state is usually interpreted as active. In the context of a gene network, the activity of a node standing for a gene represents its current transcriptional status.

A basic type of analysis that can be performed with ANIMO is the generation of a simulation run, which is presented to the user in the form of a graph (to the right). The activity graphs generated by ANIMO show the variation in activity of selected nodes over the course of the simulation run. In addition

[1] We also refer to the problem of *state space explosion*: when a model contains too many loosely coupled components, the set of its possible evolutions grows exponentially, to the point of making it impossible to effectively apply analysis techniques.

to that, a slider placed under each graph allows the user to color the nodes in the Network panel depending on their activity level at any point during the course of the simulation: the Legend panel on the left links colors to activity. ANIMO models are based on the formalism of Timed Automata [2], which is not directly shown to the user. Indeed, ANIMO was designed with the aim of enabling expert biologists to formalize their knowledge without the need for additional mathematical training. For the details on the Timed Automata model used in ANIMO, we refer the interested reader to [26].

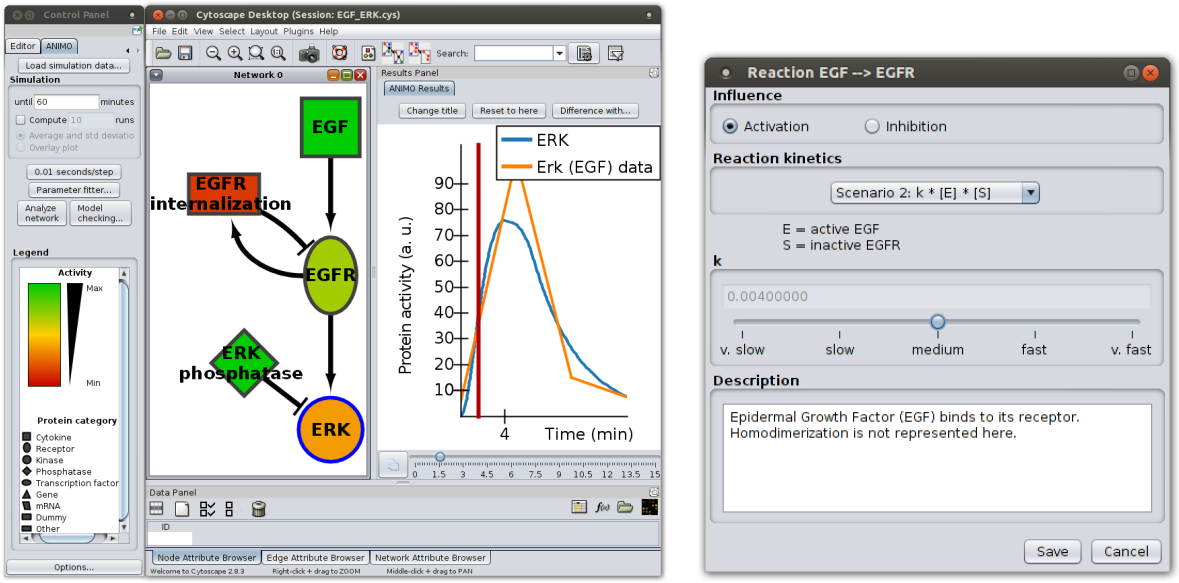


Figure 1: ANIMO user interface. (a) The main window of Cytoscape, with the ANIMO plug-in showing a network model and the result of a simulation. (b) Editing the parameters for an ANIMO interaction.

Of the two main components of an ANIMO model, only the topology is immediately visible to the user in the Network panel; the parameters are accessed by double clicking the arcs representing node-node interactions. The dialog window that is shown for an interaction contains the details of the abstract reaction kinetic describing the interaction, together with the current value of its parameter k (see Fig. 1b). The unique parameter associated to any interaction in ANIMO is used as a scale factor to make the modelled reaction occur faster or slower. We also provide the user with pre-set values for k, encouraging an initial qualitative assignment of reaction rates as "slow", "fast" and so forth. This approach is based on the assumption that biological networks are inherently robust: once an acceptable set of parameters is found, a more precise parameter search will generally have little impact on the fitness of a model to a reference data set. However, as can be seen in the small example in Figure 2, robustness does not imply that any parameter choice will do. In that example, all parameters are initially set to *medium* (Fig. 2a). The peak in EGFR was obtained setting the value of k for EGFR internalization $\dashv$ EGFR to *fast* (Fig. 2b). As this is not enough to get a peak also for ERK, which remains constantly inactive, a lower value of k for ERK phosphatase $\dashv$ ERK can be tried. A first attempt with *slow* (Fig. 2c) leads to a low peak, which can be increased by further lowering that parameter to *very slow* (Fig. 2d). Further adjustment of the parameters and non-default scenario choices (explained later) leads to the graph shown in Figure 1a. Network motifs such as the feedback loop shown in Figure 2 make models more dependent

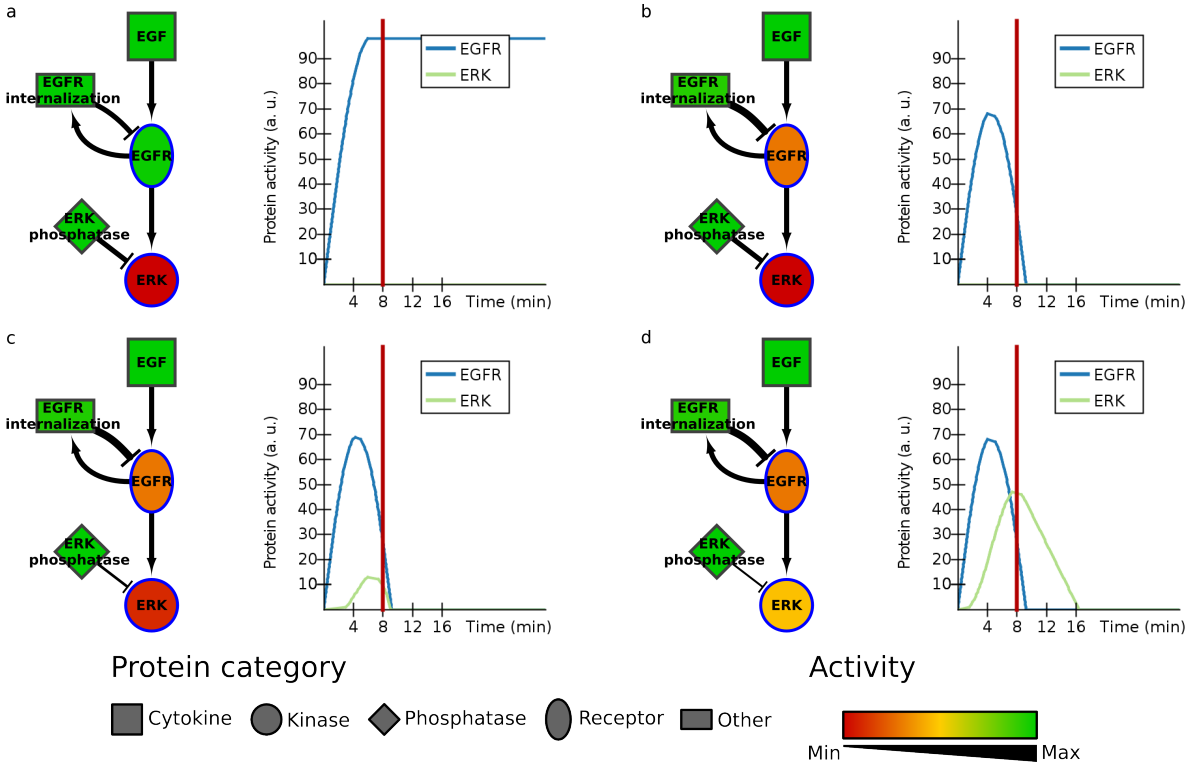


Figure 2: Example parameter settings on a simple feedback network. Arc width on the left represents parameter values: thicker arcs correspond to higher values. Colors on the left represent node activities at the points in time highlighted by the vertical red lines in the corresponding graphs on the right.

on parameter settings. As biological networks tend to heavily rely on cross-talk, network topologies can rapidly become complex. Some manual parameter fitting is currently necessary for the more complex ANIMO models. This work is sometimes slow and error-prone, taking away some of the user-friendliness for which ANIMO aims. We will show how ANIMO has been improved to make parameter choice easier for the user.

3 Support for parameter synthesis in ANIMO

When a model does not match experimental data, ANIMO offers three main tools to achieve a better fit: manual parameter editing, comparisons between different model versions, and the newly introduced parameter sweep.

3.1 Manual parameter editing

Double-clicking on an arrow in the Cytoscape representation of an interaction lets the user access a dialog that allows to change the approximated scenario and parameters for the selected interaction. The simplified scenarios were described in detail in [26], here a rough introduction is given.

In ANIMO models, the *activity level* defines the fraction of a molecular species that is considered active. The activity level is expressed in ANIMO with an integer value between 0 (completely inactive) and a maximum value that represents complete activity. The maximum value can be chosen by the user

on a range between 1 and 100, determining the granularity (or precision) with which the activity of the reactant is represented in the model. For example, an ANIMO node with 2 levels of granularity (thus with maximum activity set to 1) can be used to represent a gene, which is considered as either completely active or completely inactive. A more precise representation of the activity level is necessary when we want to represent more complex dynamics, where also intermediate values are considered important. The activity level of a node can be changed in an ANIMO model by way of activations and inhibitions, which make it respectively rise and decrease. Each occurrence of an abstract interaction changes the activity level of the downstream node by 1 step. For example, consider the interaction MEK $\rightarrow$ ERK, which represents the activation of ERK by (active) MEK. Each time that interaction occurs, the activity level of ERK rises by 1: if ERK has 100 activity levels, it will take 100 interactions to take it from full inactivity to full activity. The rate at which such interactions will occur is determined by the scenario and parameter of the interaction.

The scenarios are used to choose which are the nodes to be taken into account when computing the speed at which an interaction occurs.

Scenario 1 the simplest of the three, this scenario approximates an interaction without taking into account the abundance of substrate (what is being changed by the interaction): the activation/inhibition rate depends only on the activity level of the upstream node. In the MEK $\rightarrow$ ERK example, the rate of the interaction would be linearly dependent on the current activity level of MEK.

Scenario 2 scenario 2 models the interaction rate to be directly dependent both on the activity level of the upstream node and on the availability of substrate. By substrate we mean the inactive downstream node in case of activation interaction, and active downstream node in case of inhibition. In the MEK $\rightarrow$ ERK example, the interaction rate is linearly dependent on both the activity level of MEK and the inactive fraction of ERK. The inactive level of a node is computed by subtracting the current activity level from the maximum activity.

Scenario 3 the nodes on which this scenario depends can be chosen directly by the user, and can be the active or inactive fraction of any two nodes in the network. This scenario can be used to represent an AND-gate, where two nodes are required to be simultaneously active in order to influence their target.

The choice for the parameter k can be made directly by inserting a numeric value, or indirectly by choosing a preset among the proposed qualitative values *very slow, slow, medium, fast, very fast*.

3.2 Comparing model versions

Particularly large networks make it more difficult for the user to understand the effects of a change in network topology or interaction parameters. To overcome this difficulty, ANIMO allows the user to visually compare two versions of a model in terms of their simulation results. The Results panel in Figure 1a shows the graph of selected node activities during the course of an ANIMO simulation. The button labelled "Difference with..." allows to compare the current simulation data with another based on a possibly different version of the model. When the two simulations to be compared have been chosen, ANIMO produces a new graph plotting the difference between the two original simulations. The slider under the new graph allows to visualize also in the Network panel the changes in node activity in the whole network: Figure 3 shows the difference between the first and last version of the model in Figure 2: the difference was computed as `d - a`. Nodes colored in green in the Network panel are more active in version `d`, while red nodes are more active in version `a`.

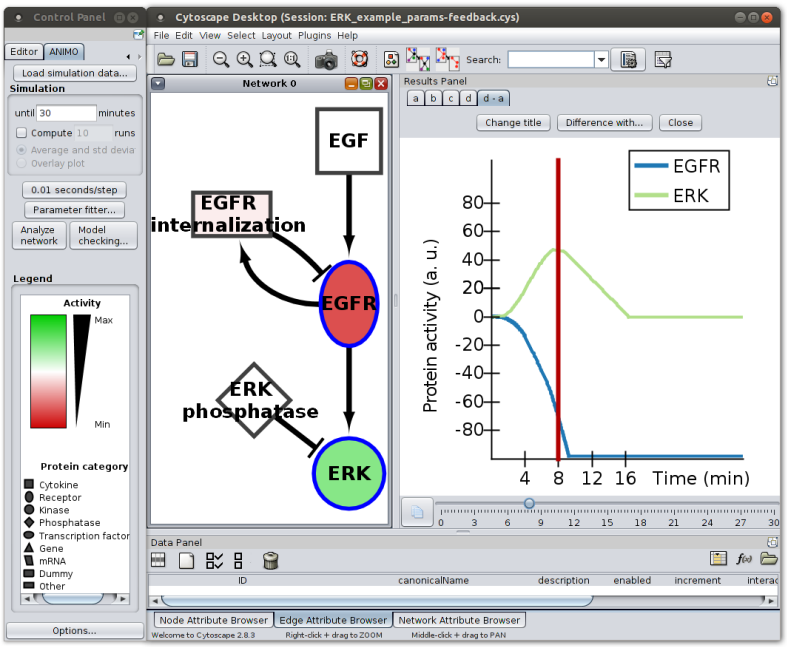


Figure 3: Difference between the graphs in Figures 2a and 2d. The Network panel shows the activity difference in the network topology at the chosen simulation point represented by a vertical red line on the graph in the Result panel.

By changing the title of a simulation, a user can keep track of different versions of their model, which can be also kept and saved with the model, to be used in future sessions or for sharing purposes. Should a user want to backtrack to a particular version of the network, the "Reset to here" button can be used (see Fig. 1a, buttons above the graph). As an additional help to recall what a model version consisted of, the tooltip of the "Reset to here" button shows an image of the network topology, taken in the moment when the selected simulation was begun. A click on the "Reset to here" button will reset the network topology and parameters at the ones used to generate the selected simulation. As the adaptation of a model may proceed on different paths, the "Difference with..." and "Reset to here" buttons provide the user with some help with tracking the changes to the model and selecting the most promising ones.

3.3 Parameter sweep

In order to avoid introducing too many changes to the topology of a network, ANIMO users should be reasonably sure that a topology does not fit a given data set before modifying the structure of the network. As this would entail an extensive parameter search which, even with the tools described in Sections 3.1 and 3.2, would require a considerable amount of time and effort, we have provided the latest version of ANIMO with a support for parameter sweeps. The idea is to let ANIMO automatically explore a user-defined parameter space by computing a series of simulations based on a constant network topology, where only the parameters are being changed. Comparing the results with a given experimental data series will allow for an automatic pre-screening, leaving the user with a choice among the best fitting settings. Moreover, the task of parameter sweeping can be trivially parallelized, allowing us to exploit the multi-core architectures available in most personal computers, and present the users with the requested results in considerably less time. In our case, the multi-core parameter sweep is implemented by using a thread pool pattern [14], relying on the local scheduler to have the threads distributed on all CPU cores.

Another important option provided in ANIMO is the choice between linear and logarithmic scales when defining the way a parameter space has to be explored: this gives the user more control on the balance between precision and performances. When considering a big set of interactions or of parameter settings, it is faster to proceed in logarithmic steps. After a more precise interval of parameters has been identified for a subset of the interactions, a linear search becomes more feasible. In this way, the computational resources can be used more efficiently and results can be obtained faster.

The goodness of a model configuration is determined in ANIMO by comparing the activity graph resulting from a simulation with the selected configuration against a given experimental data set, applying the following formula for each of the selected data series i:

$$Error_i = \max_{t=0}^{maxTime} \frac{|Data_i(t) - Model_i(t)|}{nLevels_i}$$

where $Data_i(t)$ is the experimental data point at time t for the series i, $Model_i(t)$ is the corresponding value on the graph computed by ANIMO with the current parameter configuration, and $nLevels_i$ is the granularity (number of levels) of the network node i. The error of the given model configuration is then computed as the maximum error over all selected data series:

$$Error = \max_{i=0}^{nSeries} (Error_i)$$

Thanks to the support for parameter sweeps, ANIMO users can obtain better insight on the parameter sensitivity of their models, identifying critical points in the networks. This can help identifying the most sensitive areas of a network, and possibly suggest some intervention points for successive topology expansions, if model robustness needs to be enforced. Note that the approach is only sketched here: statistical considerations on confidence should also be taken into account.

All the simulations computed in ANIMO when performing a parameter sweep are based on a deterministic version of the model: a given parameter configuration gives always the same simulation as result. This is why we use only one simulation per configuration when comparing the results with experimental data. However, the user can add some non-determinism to the model by defining uncertainty intervals around which interaction times are distributed. For example, the default setting of 5% uncertainty illustrated in [26] implies that an interaction can complete in a time t sampled from an uniform distribution in the continuous interval $[0.95 \times T, 1.05 \times T]$. Here, T is the exact time for one interaction step, and is computed by applying the selected scenario to the current activity levels of the nodes involved in the interaction. Adding uncertainty to a model can be used to test its robustness. The user can then ask ANIMO to compute a number of simulations in *overlay* mode, which will allow the user to see all the simulations superposed in one graph, and realize whether in some cases the model significantly deviates from its normal behavior.

We now present an example application of ANIMO's parameter sweep feature on the model shown in Figure 1a. Thanks to parameter sweep, we can make a more exhaustive search than the one presented in Figure 2, using an automatic approach instead of a manual trial-and-error process. We started by choosing scenario 2 for the two interactions involving ERK, while the other three interactions were left to scenario 1: this allows us to obtain a smoother looking graph for ERK. A logarithmic parameter sweep was performed, letting the parameter values vary over the 5 pre-sets "very slow" ($k = 0.001$), "slow" ($k = 0.002$), "medium" ($k = 0.004$), "fast" ($k = 0.008$), "very fast" ($k = 0.016$) for all 5 interactions in the network. This resulted in $parameter\ choices^{interactions} = 5^5 = 3\,125$ simulations, that were computed in 43 seconds on an eight-core Intel® Core™ i7 CPU at 2.80GHz equipped with 4 Gb RAM and running Ubuntu GNU/Linux 13.10 64bit. The model we consider here is small enough for one simulation to take

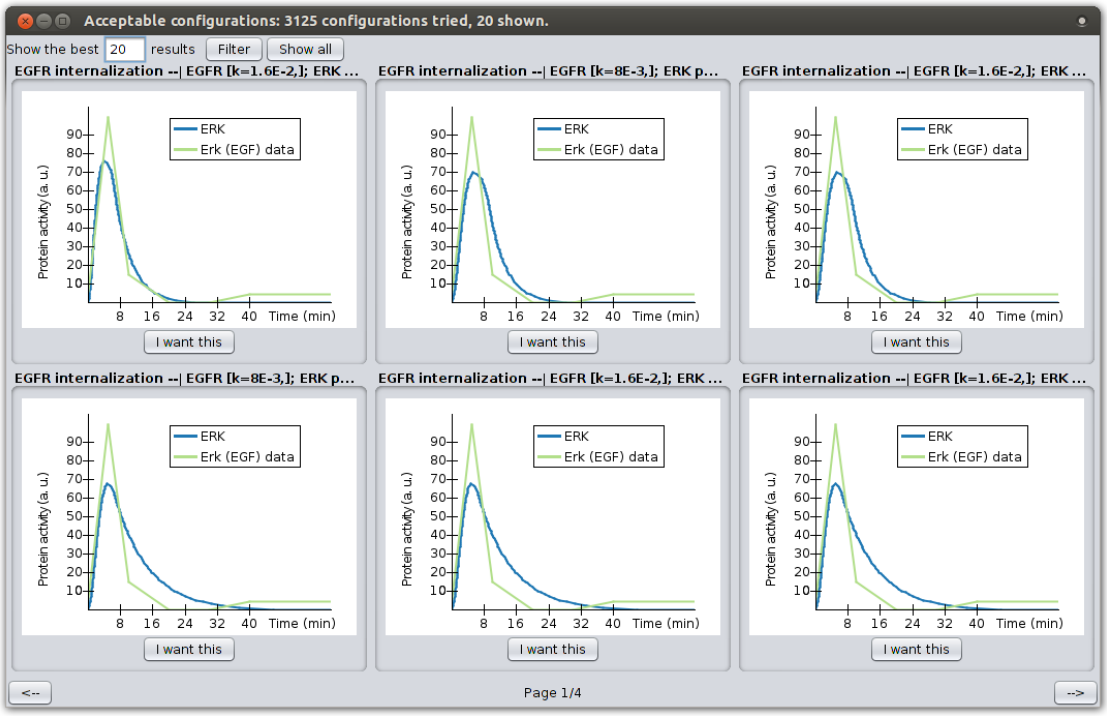


Figure 4: Result window of a logarithmic parameter sweep in the ANIMO model of Figure 1a.

Reaction	k
EGF $\rightarrow$ EGFR	Fast
EGFR $\rightarrow$ EGFR internalization	Medium
EGFR internalization $\dashv$ EGFR	Very Fast
EGFR $\rightarrow$ ERK	Fast
ERK phosphatase $\dashv$ ERK	Medium

Table 1: Qualitative values of the parameters k used in the example model to obtain the graph shown in Figure 1a.

less than a second, but the exponential growth of the number of simulations required for a parameter sweep can make the sheer number of tasks challenging for any processing unit. For example, a more complex model involving 10 interactions instead of 5 would require $5^{10} = 9\,765\,625$ simulations, which could be computed in approximately a day and a half on an eight-core machine. Still, the possibility to compute simulations in parallel provides a significant help in reducing the impact of an exponential growth of the space to be explored.

The result window shown by ANIMO at the end of the computation of the 3 125 simulations can be seen in Figure 4, where the best 6 results are listed in form of graphs. By interacting with the interface, the user can choose to see more of the top-scoring results: this will allow to check the width of the parameter settings for which the wanted behavior can still be considered "close enough" to the experimental data. Pressing one of the buttons labelled "I want this" under one of the graphs allows the user to copy the parameters used in the selected simulation to the network model. The graph in Figure 1a was obtained by choosing the first graph shown in Figure 4, i.e. setting the parameters as in Table 1. It can be seen that the model does not fit the data very precisely: this is because some nodes known to influence the activity of ERK were purposely left out of the example for simplicity's sake.

4 Suggested ANIMO workflow

In order to better profit from what ANIMO offers to the biologist, we suggest a step-by-step modeling workflow:

1. Define a starting topology, based on literature data and current knowledge on the network.
2. Choose the interaction parameters from the pre-set qualitative values, based on experience (e.g., protein expression is much slower than phosphorylation). The choice on the approximated kinetics can be for the most part based on scenario 1, referring to Section 3.1.
3. Check that the network behaves as expected (e.g. addition of EGF makes ERK activity rapidly increase and successively decrease).
4. Possibly iterate the steps 1 to 3 until the network behaves as expected.
5. Compare the model with experimental data.
6. If the model does not fit the data, change the parameters using more precise numerical values and/or scenario settings.
7. If the data fitting is still not satisfactory, apply parameter sweep to (subsets of) the model as described in Section 3.3.
8. When a good fit is found, add non-determinism to the model and ensure that the behavior does not change significantly under reasonably high uncertainty settings.
9. If the model can fit the data only with very precise parameter settings, or a parameter configuration that fits the data cannot be found after a reasonably extensive search on the parameter space, change the network topology possibly adding more nodes from literature. If no candidates can be found in literature, speculative candidates can be found by "playing" with the current configuration in ANIMO's user interface. We advise to let expert biologists perform this step on their own, or to perform it under their supervision, in order to keep the network both simple and realistic. For the same reason, we also advise not to introduce more than a couple of nodes at a time.
10. Parameters for newly introduced interactions can be chosen as in step 2, possibly iterating steps 2 to 7 to refine the parameters.
11. To check how changes in the network are reflected in the behavior of the model, we advise to use the comparison tools described in Section 3.2.

If the model fits the data reasonably well only after modifications to the initial topology, it may contain some hypothetical or unexpected nodes. In that case, additional laboratory experiments can be designed starting from the hypotheses introduced in the model, and help the biologists decide which are the most precise explanations for the experimental observations.

A simplified graphical representation of the workflow can be found in Figure 5: the three degrees of precision with the parameter settings are highlighted with progressively darker shades of blue. Moreover, the figure helps to stress the fact that in our method changes in topology (green boxes) should be made as a last step, after a reasonable attempt at making the model fit via parameter adjustments. Again, it is important to remember that if a model behaves correctly only for very narrow parameter settings then it may not be as robust as expected, so a review of its topology may be needed.

The process described above allows the biologists to formalize their knowledge, and makes it easier to have group discussions when trying to combine multiple subnetworks in a more comprehensive model. It is worth noting that experience in both the specific biological setting and generic modeling of biological

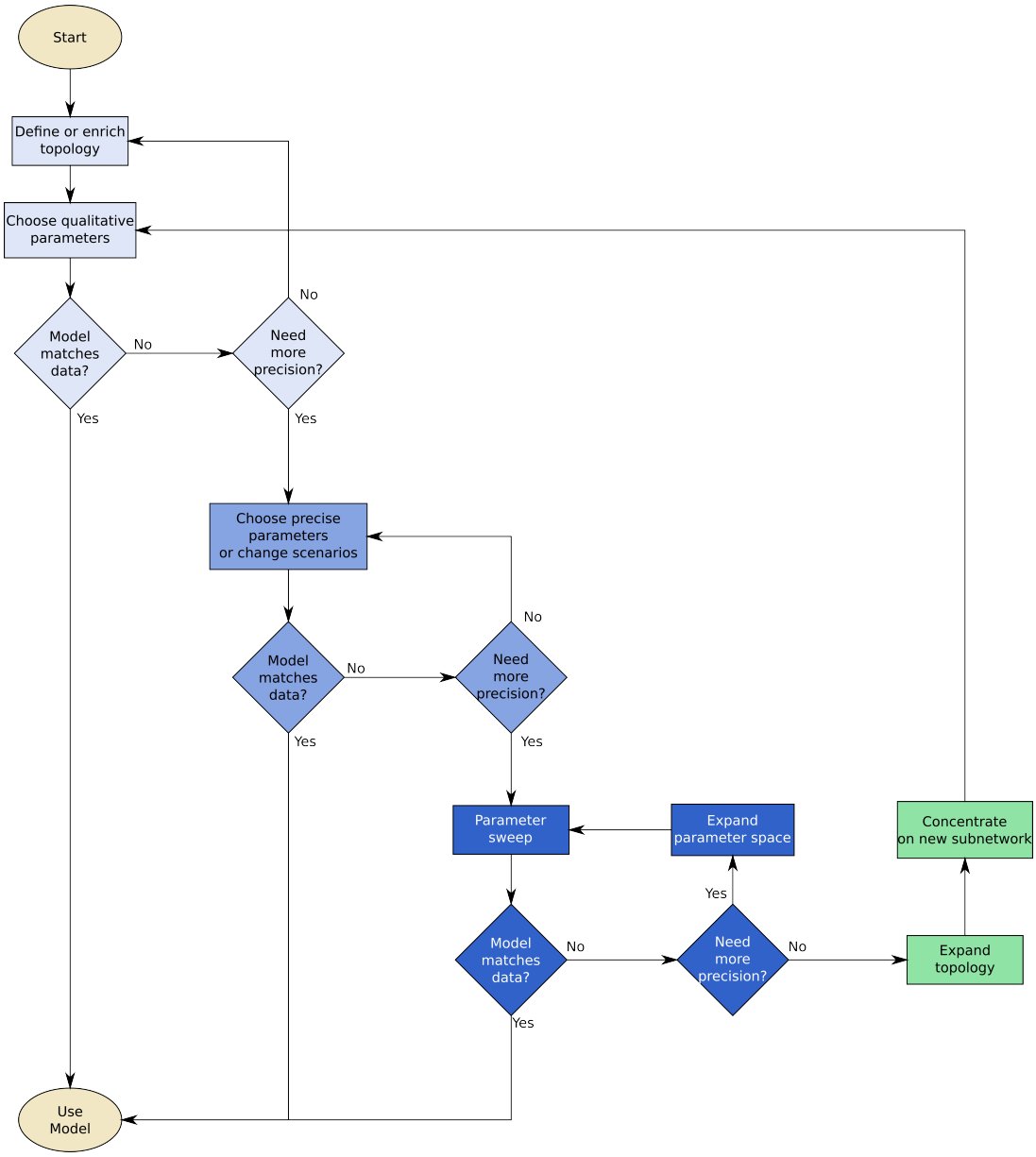


Figure 5: ANIMO modeling workflow.

events represents an important asset for a faster and more realistic modeling process with ANIMO. However, as we claimed with all previous versions of the tool, it is still the case that no experience in formal methods is needed in order to use ANIMO. In particular, an ideal ANIMO user could have no experience on the Timed Automata foundations of the tool and yet fully profit from its features.

5 Conclusions

ANIMO has been extended with a proper support for parameter synthesis. Now the tool can be used more efficiently to model biological networks, without concentrating exclusively on their topology. In particular, the addition of an automatic parameter sweep feature allows the user to save considerable amounts of time, keeping closer to experimental data without the need to perform a parameter search by hand.

In the future, we plan to improve our parameter analysis techniques, possibly using some of the techniques already developed for testing parameter sensitivity [3, 13] and robustness [5, 9, 18, 30] in Timed Automata and hybrid systems more in general [23, 4]. Techniques adopted in other formalisms such as e.g. differential equations [22, 11, 19], rule-based languages [6, 28] or Bayesian networks [21] are also being considered as a source of inspiration. As already stated, parameter sensitivity is very important in biological models: if a treatment derived from a model is based on narrow bounds for a parameter, that parameter must not exhibit significant variety among patients. By introducing an automatic analysis for parameter sensitivity, our tool could point out the zones of a network model where the user should concentrate more, for example by improving the topology, or by performing more focused parameter sweeps. Moreover, we also aim at applying some results from automata learning [29] to the generation of Timed Automata models based on experimental data: generating models that exhibit specific behaviors implies a careful definition of their parameter space. Finally, parametric model checking [15] represents an interesting development direction: having a tool automatically output the parameter intervals for which some given outputs are observed is a highly valued resource for the biological researcher.

References

[1] Maximino Aldana & Philippe Cluzel (2003): *A natural class of robust networks*. Proceedings of the National Academy of Sciences 100(15), pp. 8710–8714, doi:10.1073/pnas.1536783100.

[2] Rajeev Alur & David L. Dill (1994): *A Theory of Timed Automata*. Theor. Comput. Sci. 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[3] Étienne André, Thomas Chatain, Laurent Fribourg & Emmanuelle Encrenaz (2008): *An Inverse Method for Parametric Timed Automata*. Electronic Notes in Theoretical Computer Science 223(0), pp. 29 – 46, doi:10.1016/j.entcs.2008.12.029.

[4] Étienne André (2010): *IMITATOR II: A Tool for Solving the Good Parameters Problem in Timed Automata*. In: INFINITY, pp. 91–99, doi:10.4204/EPTCS.39.7.

[5] Patricia Bouyer, Nicolas Markey & Pierre-Alain Reynier (2006): *Robust Model-Checking of Linear-Time Properties in Timed Automata*. In: LATIN, pp. 238–249, doi:10.1007/11682462_25.

[6] Nathalie Chabrier-Rivier, Franois Fages & Sylvain Soliman (2005): *The Biochemical Abstract Machine BIOCHAM*. In Vincent Danos & Vincent Schachter, editors: Computational Methods in Systems Biology, Lecture Notes in Computer Science 3082, Springer Berlin Heidelberg, pp. 172–191, doi:10.1007/978-3-540-25974-9_14.

[7] Claudine Chaouiya, Elisabeth Remy, Brigitte Mossé & Denis Thieffry (2003): *Qualitative Analysis of Regulatory Graphs: A Computational Tool Based on a Discrete Formal Framework*. In Luca Benvenuti, Alberto De Santis & Lorenzo Farina, editors: *Positive Systems*, *Lecture Notes in Control and Information Sciences* 294, Springer, Berlin / Heidelberg, pp. 830–832, doi:10.1007/978-3-540-44928-7_17.

[8] Federica Ciocchetta, Adam Duguid, Stephen Gilmore, Maria Luisa Guerriero & Jane Hillston (2009): *The Bio-PEPA Tool Suite*. *International Conference on Quantitative Evaluation of Systems*, pp. 309–310, doi:10.1109/QEST.2009.27.

[9] Conrado Daws & Piotr Kordy (2006): *Symbolic Robustness Analysis of Timed Automata*. In: *FORMATS*, pp. 143–155, doi:10.1007/11867340_11.

[10] Lorenzo Dematté, Corrado Priami & Alessandro Romanel (2008): *Modelling and simulation of biological processes in BlenX*. *SIGMETRICS Perform. Eval. Rev.* 35, pp. 32–39, doi:10.1145/1364644.1364653.

[11] Alexandre Donzé (2010): *Breach, a Toolbox for Verification and Parameter Synthesis of Hybrid Systems*. In: *Proceedings of the 22Nd International Conference on Computer Aided Verification*, CAV'10, Springer-Verlag, Berlin, Heidelberg, pp. 167–170, doi:10.1007/978-3-642-14295-6_17.

[12] Jasmin Fisher & Thomas A. Henzinger (2007): *Executable cell biology*. *Nature Biotechnology* 25(11), pp. 1239–1249, doi:10.1038/nbt1356.

[13] Laurent Fribourg & Ulrich Kühne (2011): *Parametric Verification and Test Coverage for Hybrid Automata Using the Inverse Method*. In Giorgio Delzanno & Igor Potapov, editors: *Reachability Problems*, *LNCS* 6945, Springer Berlin Heidelberg, pp. 191–204, doi:10.1007/978-3-642-24288-5_17.

[14] Rajat P. Garg & Ilya Sharapov (2002): *Techniques for Optimizing Applications: High Performance Computing*. Prentice Hall Professional Technical Reference.

[15] Thomas Hune, Judi Romijn, Mariëlle Stoelinga & Frits Vaandrager (2002): *Linear parametric model checking of timed automata*. *The Journal of Logic and Algebraic Programming* 52-53(0), pp. 183 – 220, doi:10.1016/S1567-8326(02)00037-1.

[16] Hidde de Jong, Johannes Geiselmann, Céline Hernandez & Michel Page (2003): *Genetic Network Analyzer: qualitative simulation of genetic regulatory networks*. *Bioinformatics* 19(3), pp. 336–344, doi:10.1093/bioinformatics/btf851.

[17] Sarah Killcoyne, Gregory W. Carter, Jennifer Smith & John Boyle (2009): *Cytoscape: a community-based framework for network modeling*. *Methods in molecular biology (Clifton, N.J.)* 563, pp. 219–239, doi:10.1007/978-1-60761-175-2_12.

[18] P. T. Kordy, R. Langerak & J. W. Polderman (2010): *Re-verification of a Lip Synchronization Protocol using Robust Reachability*. *Electronic Proceedings in Theoretical Computer Science* 20, pp. 49–62, doi:10.4204/EPTCS.20.5.

[19] Paola Lecca, Alida Palmisano, Adaoha Ihekwaba & Corrado Priami (2010): *Calibration of dynamic models of biological systems with KInfer*. *European Biophysics Journal* 39(6), pp. 1019 – 1039, doi:10.1007/s00249-009-0520-3.

[20] Petros Lenas, Malcolm Moos Jr. & Frank P. Luyten (2009): *Developmental Engineering: A New Paradigm for the Design and Manufacturing of Cell-Based Products. Part II. From Genes to Networks: Tissue Engineering from the Viewpoint of Systems Biology and Network Science*. *Tissue Engineering Part B: Reviews* 15(4), pp. 395–422, doi:10.1089/ten.teb.2009.0461.

[21] Wenhui Liao & Qiang Ji (2009): *Learning Bayesian network parameters under incomplete data with domain knowledge*. *Pattern Recognition* 42(11), pp. 3046 – 3056, doi:10.1016/j.patcog.2009.04.006.

[22] Pedro Mendes, Stefan Hoops, Sven Sahle, Ralph Gauges, Joseph Dada & Ursula Kummer (2009): *Computational modeling of biochemical networks using COPASI systems biology*. In Ivan V. Maly, John M. Walker & John M. Walker, editors: *Systems Biology*, chapter 2, *Methods in Molecular Biology* 500, Humana Press, Totowa, NJ, pp. 17–59, doi:10.1007/978-1-59745-525-1_2.

[23] Joseph S. Miller (2000): *Decidability and Complexity Results for Timed Automata and Semi-linear Hybrid Automata*. In: *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control*, HSCC '00, Springer-Verlag, London, UK, pp. 296–309, doi:10.1007/3-540-46430-1_26.

[24] Masao Nagasaki, Ayumu Saito, Euna Jeong, Chen Li, Kaname Kojima, Emi Ikeda & Satoru Miyano (2011): *Cell illustrator 4.0: a computational platform for systems biology.* *Stud Health Technol Inform* 162, pp. 160–81.

[25] Corrado Priami (2009): *Algorithmic systems biology*. *Commun. ACM* 52, pp. 80–88, doi:10.1145/1506409.1506427.

[26] Stefano Schivo, Jetse Scholma, Brend Wanders, Ricardo A. Urquidi Camacho, Paul E. van der Vet, Marcel Karperien, Rom Langerak, Jaco van de Pol & Janine N. Post (2013): *Modelling biological pathway dynamics with Timed Automata*. *IEEE Journal of Biomedical and Health Informatics* In press, doi:10.1109/JBHI.2013.2292880.

[27] Jetse Scholma, Stefano Schivo, Ricardo A. Urquidi Camacho, Jaco van de Pol, Marcel Karperien & Janine N. Post (2014): *Biological networks 101: Computational modeling for molecular biologists*. *Gene* 533(1), pp. 379–384, doi:10.1016/j.gene.2013.10.010.

[28] Adam Smith, Wen Xu, Yao Sun, James Faeder & G Elisabeta Marai (2012): *RuleBender: integrated modeling, simulation and visualization for rule-based intracellular biochemistry*. *BMC Bioinformatics* 13(Suppl 8), p. S3, doi:10.1186/1471-2105-13-S8-S3.

[29] Jan Tretmans (2011): *Model-Based Testing and Some Steps towards Test-Based Modelling*. In Marco Bernardo & Valérie Issarny, editors: *Formal Methods for Eternal Networked Software Systems*, *Lecture Notes in Computer Science* 6659, Springer, Berlin / Heidelberg, pp. 297–326, doi:10.1007/978-3-642-21455-4_9.

[30] Martin Wulf, Laurent Doyen, Nicolas Markey & Jean-François Raskin (2008): *Robust safety of timed automata*. *Form. Methods Syst. Des.* 33(1-3), pp. 45–84, doi:10.1007/s10703-008-0056-7.

Toward Parametric Timed Interfaces for Real-Time Components

Youcheng Sun[1,2], Giuseppe Lipari[1,2], Étienne André[3] and Laurent Fribourg[2]
[1]Scuola Superiore Sant'Anna, Pisa, Italy*
[2]LSV, ENS Cachan & CNRS, France
[3]Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France†

We propose here a framework to model real-time components consisting of concurrent real-time tasks running on a single processor, using parametric timed automata. Our framework is generic and modular, so as to be easily adapted to different schedulers and more complex task models. We first perform a parametric schedulability analysis of the components using the inverse method. We show that the method unfortunately does not provide satisfactory results when the task periods are considered as parameters. After identifying and explaining the problem, we present a solution adapting the model by making use of the worst-case scenario in schedulability analysis. We show that the analysis with the inverse method always converges on the modified model when the system load is strictly less than 100%. Finally, we show how to use our parametric analysis for the generation of timed interfaces in compositional system design.

Keywords: Real-Time Scheduling, Parametric Schedulability Analysis, Parametric Timed Automata.

1 Introduction

Designing and analysing distributed real-time systems is a very challenging task. The main source of complexity arises from the large number of parameters to consider: tasks priorities, computation times and deadlines, synchronisation, precedence and communication constraints, etc. Finding the optimal values for the parameters is not easy and often a small change in one parameter may completely change the behaviour of the system and even compromise its correctness. For these reasons, designers are looking for analysis methodologies that allow incremental design and exploration of the parameter space.

We consider here real-time systems consisting of a set of real-time tasks executed concurrently on a single processor platform. Each task can be time-triggered or event-triggered: in the first case, it is activated periodically, and each time it executes a portion of code called *job* or *instance*, after which it self-suspends, waiting for their next periodic activation. In the second case, instances are activated by internal or external events. Each task is characterised by a *relative deadline*, that is the maximum amount of time that must elapse from the activation of one instance to its completion.

A *scheduler* is needed to decide which task to execute at each instant. The scheduler can be *on-line* if the decision is taken while the system is running depending on the current state; or *off-line* if the schedule is pre-computed before the system runs. Fixed Priority Preemptive Scheduling (FPPS) has been standardised in POSIX [1] and is currently available in all commercial and open-source Real-Time Operating Systems.

*The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 246556.

†This work is partially supported by STIC Asie project CATS ("Compositional Analysis of Timed Systems").

É. André and G. Frehse (Eds.): 1st International Workshop on Synthesis of Continuous Parameters (SynCoP'14).
EPTCS 145, 2014, pp. 49–64, doi:10.4204/EPTCS.145.6

One important requirement of real-time systems is to ensure that the system is *schedulable*, i.e. that all tasks will always complete before their deadlines when scheduled by the selected algorithm. Testing the system under different input and state conditions does not guarantee the system *schedulability* (i.e. that the system is schedulable), because the number of possibilities to test is too large to guarantee complete coverage of all possible cases. A better approach is to build an abstract model of the system, and perform analysis on the model.

A large body of research literature has addressed the problem of schedulability analysis of real-time tasks, both using formal methods (e.g. [2, 15, 16]) and mathematical equations (e.g. [12, 25]). In the literature, a task is typically modelled by several parameters, typically (i) a *worst-case computation time* (i.e. an upper bound of the execution time of every instance of the task under every possible condition), (ii) a *deadline*, and (iii) an *activation pattern* (e.g. periodic, sporadic, arrival curve). A periodic task is activated every period; a sporadic task can be activated at any time, but the distance between two activations is lower bounded by a constant *minimum interarrival time*; finally, an arrival curve [28] is a function $\alpha(t)$ that defines an upper bound on the maximum number of activations in any interval of length t.

Real-Time Components and Timed Interfaces

For complex distributed real-time systems, a component-based methodology may help reduce the complexity of the design and analysis phases. This paper is a first step toward the definition of a *timed interface* for a real-time component. Therefore, we now describe our notion of real-time component and timed interface.

We define a *distributed real-time system* as a set of *real-time components*. Each component runs on a dedicated single processor node, and all components are connected to each other by a local network. A component consists of a *provided interface*, a *required interface*, and an *implementation* (see e.g. [17]).

The *provided interface* is a set of methods that a component makes available to other components of the system. Each method is characterised by: (i) the method signature, which is the name of the method and the list of parameters, and (ii) a worst-case activation pattern, which describes the maximum number of invocations the method is able to handle in any interval of time. In this paper, we will describe the worst-case activation pattern by an *arrival curve* [28]. The semantic of invocation of a method can be synchronous (the caller waits for the method to be completed) or asynchronous (the caller continues to execute without waiting for the completion of the operation).

The *required interface* is a set of methods that the component requires for carrying out its services. Each method is characterised by its signature and a worst-case invocation pattern.

The *implementation* of a component is the specification of how the component carries out its work. In our model, a component is implemented by a set of concurrent real-time tasks and by a scheduler. Tasks can be time-triggered, when periodically activated; or event-triggered, in which case they are activated by a call to a provided method of the component. In other words, an event-triggered task implements one method of the provided interface, and in turn it may invoke a method of the required interface.

A graphical representation of a component is shown in Figure 1. In this example, the component provides one single method in the provided interface (pictorially represented by the red rectangle), and does not specify any method in the required interface. The component is implemented by three tasks: tasks τ_1 and τ_3 are time triggered (the green clocks in the picture), whereas task τ_2 implements the method in the provided interface, and hence it is triggered by invocations from external clients.

In a *component-based* design methodology, components are independently designed and developed, and then *integrated* in the final system by connecting them together through their interfaces. It is clear

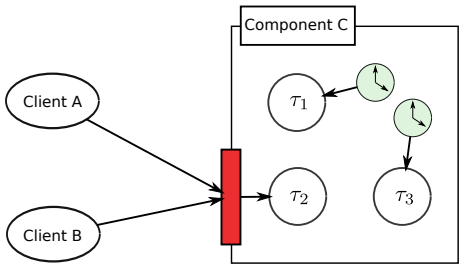


Figure 1: A component with three tasks and one method in the provided interface.

that the interface specification plays an important role in this methodology: for a real-time component, the interface should contain not only the functional specification (i.e. method signature, constraints on the parameters, etc.) but also the *timed behaviour* of the component. In particular, in this paper we enhance the specification of the interface by adding parameters on the activation pattern and on the response delay of a method.

Given a component $\mathscr{C}$, its *provided interface* is thus defined as:

- a set of method signatures $m_1, m_2, \ldots$;
- a *parametric arrival curve* $\alpha_i(t)$ for each method m_i, which represents the activation pattern that the corresponding implementing task will receive;
- a *worst-case response time* D_i parameter for each method m_i.

Similarly, the *required interface* of a component is defined as:

- a set of method signatures $m_1, m_2, \ldots$;
- a *parametric arrival curve* $\alpha_i(t)$ for each method m_i that represents the activation pattern generated by this component;
- for every synchronous method call, a maximum allowed delay R_i in receiving the response.

Finally, the component is characterised by a *set of constraints* on the parameters: for all valuations of the parameters satisfying the constraints, the component is guaranteed to be correct both from the functional point of view (i.e. the component produces correct values) and from the timing point of view (i.e. all tasks complete before their deadlines, and all provided functions return their values within the desired maximum response delay).

The road to realise such a component-based design methodology is long and many theoretical and practical problems need to be solved before the methodology can be used in practice. One important problem is how to compute the set of constraints that define the correct behaviour of a component. In the process of designing and analysing a component in isolation, it is necessary to use parametric arrival curves for describing the activation patterns for event-triggered tasks, and parametric deadlines for bounding their response times. Performing a parametric analysis aims at deriving a set of constraints for these parameters that make the component schedulable. During integration, the correctness of the system is checked by intersecting the constraints of the communicating components to see if there is some feasible assignment of parameters that makes all components schedulable.

Objectives

Our general research agenda, that goes beyond the scope of this paper, is to establish a component-based design methodology and analysis for real- time components. One of the important steps in the proposed

methodology is to be able to perform parametric analysis of a component with respect to its activation patterns.

In this paper, we focus on solving two specific problems:

1. how to build a formal parametric model of a component consisting of a set of real-time tasks, some of which can be periodic, others can be activated by generic arrival curves; and
2. how to perform a parametric analysis of the schedulability of the component, thus deriving a set of constraints that define the space of parameters that make the component schedulable.

For the sake of simplicity, in this paper we focus only on the *provided interface* of a component; that is, we investigate on the parametric analysis of a component with respect to the patterns of activations. The analysis of the required interface is the subject of future work.

Contributions

Our contribution is threefold. First, we propose a formal model of a real-time component based on parametric timed automata [4], a popular formalism for modelling real-time systems. Unlike many similar models proposed in the literature, our modelling framework is completely modular: a system is obtained by combining simpler automata, each one implementing one aspect of the component. In particular, we separate the specification of the task behaviour from the activation pattern (periodic, sporadic or generic arrival curve), and from the scheduler. In this way we can easily and seamlessly change the scheduler and the activation pattern of a task without changing the rest of the component specification, which is very important during the parametric analysis of a component. For the analysis, we use the inverse method [8] and the IMITATOR tool [7].

As a second contribution, we show that, when the activation patterns are parametric, the inverse method does not provide satisfactory results, in the sense that it may output a constraint reduced to a single point. We describe the problem and provide a solution that is valid for periodic (with no offset), sporadic and generic activation patterns that can be described by arrival curves.

Finally, as third contribution, we describe how this model can be used as a basis for synthesising the timed interface of a real-time component.

Organisation of the Paper

The rest of this paper is organised as follows. Section 2 reviews related work. Section 3 recalls the necessary preliminaries, viz. real-time systems, parametric timed automata and the inverse method. Section 4 presents our model of a real-time component using parametric timed automata. Section 5 introduces our parametric analysis, allowing to deal with parametric task activations. Section 6 introduces preliminary work allowing to perform a component-based parametric analysis. Section 7 concludes and present further directions of research.

2 Related Work

Parametric analysis of real-time systems using mathematical equations has already been addressed in the past. Bini et al. [10] proposed a method for parametric analysis of real-time periodic tasks where parameters can be either worst-case computation times or task periods. However, with Bini et al.'s approach, changing the task model requires the development of a new methodology.

A more general approach to scheduling analysis is to use formal methods for modelling a real-time system. A formal framework for scheduling problems using timed automata with stopwatches has been proposed in [2]. Fersman et al. [15] proposed a *Task Automaton*. Similar approaches have been proposed using time(d) Petri nets [11, 22].

It is possible to perform an exploration of the parameter space using timed automata, as in [20]. However, their approach is not fully parametric: the analysis is repeated for all combination of the discrete values of the parameters. Hence, their method does not scale well as the number of parameters and the number of discrete values increases. Furthermore, that approach does not consider non-integer points, and cannot be used to quantify the system robustness.

Full parametric analysis can be performed using specific formalisms. For example, formalisms such as parametric timed automata (PTA) [4] and parametric time Petri nets [29], have been used to model parametric schedulability problem (see, e.g. [14, 27]). In particular, thanks to generality of these modelling languages, it is possible to model a larger class of constraints, and perform full parametric analysis on many different variables, for example task offsets.

The inverse method IM [8] can be used for exploring the space of parameters of a parametric timed automaton (and, more generally, of a parametric *stopwatch* automaton) in the proximity of a valuation point. In this paper we use this method for performing parametric analysis of real-time systems where task activation patterns are modelled with parametric *arrival curves*.

We have used a similar approach in [27], where a distributed real-time system has been modelled using parametric stopwatch automata. However, in [27] the methodology is limited to only use the tasks' computation times as parameters. Here, we investigate a situation where arrival curves are considered as parameters too. Furthermore, our final goal is to be able to perform interface-based parametric analysis.

Our generic modular approach can be seen as a contract-based methodology where "provided" and "required" interfaces are instances of (assumption, guarantee) pairs in the contract terminology. An interface-based approach to the design and analysis of real-time systems using assume/guarantees has already been proposed in the literature [19, 26], but their approach is not parametric. Compositional verification of timed systems, using assume guarantee reasoning, has also been considered in [23] for event-recording automata, a subclass of timed automata; again, this approach is non-parametric.

3 Preliminaries

3.1 Real-Time Tasks

A real-time task τ_i is a sequence of instances (or jobs) $J_{i,k}$, with $k = 0, 1, \ldots$. Each instance $J_{i,k} = (a_{i,k}, c_{i,k}, d_{i,k})$ is characterised by an arrival time $a_{i,k}$, a computation time $c_{i,k}$, and an absolute deadline $d_{i,k}$. The system is schedulable if the scheduling algorithm orders the execution times of the jobs such that each job executes $c_{i,k}$ units of execution in interval $[a_{i,k}, d_{i,k}]$. Additionally, an instance can only start executing after the previous instances from the same task have completed: if we denote by $f_{i,k-1}$ the finishing time of the $(k-1)$th instance, then each job can only execute in interval $[\max(f_{i,k-1}, a_{i,k}), d_{i,k}]$.

Task τ_i is then characterised by three parameters:

- the Worst-Case Execution Time C_i, which is an upper bound on the execution time of any instance of the task (i.e. $\forall k > 0 : c_{i,k} \leq C_i$);
- the *relative deadline* D_i; the absolute deadline of every instance can be computed as $d_{i,k} = a_{i,k} + D_i$;
- the arrival pattern.

For the arrival pattern, we consider three kinds of schemes:

- **Periodic**: this arrival pattern is characterised by a *period* T_i, and the arrival time of every instance is computed as:

$$a_{i,0} = 0$$
$$\forall k > 0 : a_{i,k} = a_{i,k-1} + T_i$$

- **Sporadic**: this arrival pattern is characterised by a *minimum interarrival time* that we denote again by T_i, and the arrival times of every instance must respect the following constraints:

$$a_{i,0} = 0$$
$$\forall k > 0 : a_{i,k} \geq a_{i,k-1} + T_i$$

- **Arrival curve** [28]: in this case the pattern of arrival must respect a certain function called *arrival curve* $\alpha_i(t) : \mathbb{R} \to \mathbb{N}$. The arrival curve constrains the number n of arrivals in any interval of a given length Δ:

$$\forall k \geq 0, \forall n > 0 : n \leq \alpha_i(a_{i,k+n-1} - a_{i,k})$$

  In other words, the number of arrival events in any interval must not exceed the value of the arrival curve for that interval[1]. Arrival curves are monotonically non-decreasing, and convex, i.e. $\forall t, \delta : \alpha_i(t+\delta) \leq \alpha_i(t) + \alpha_i(\delta)$. The value of the an arrival curve at time 0 is also called *burstiness* and represents the amount of simultaneous arrival events that can be sent to a task. Arrival curves are a generalisation of the sporadic arrival model. In fact, a sporadic task can be represented by an arrival curve with burstiness $\alpha_i(0) = 1$ and a periodic behaviour. However, an arrival curve can have any convex shape.

  The sum of two arrival curves is still an arrival curve. Also, we can define a partial order relationship between arrival curves using the natural ordering between values of the function: $\alpha_i(\cdot) \preceq \alpha_j(\cdot)$ iff $\forall t\ \alpha_i(t) \preceq \alpha(t)$.

  In this paper we deal with parametric arrival curves. In particular, we will use periodic arrival curves of the form:

$$\alpha_{N^u,P}(t) = N^u + \left\lfloor \frac{t}{P} \right\rfloor \tag{1}$$

  where N^u is a discrete parameter that denotes the initial burstiness, and P is a continuous parameter that denotes the period. Using the partial order relationship, a generic arrival curve can always be upper bounded by a periodic arrival curve of the form (1).

3.2 Parametric Stopwatch Automata

We introduce here an extension of parametric timed automata that will be used in Section 4 to model real-time systems. Timed automata are finite-state automata augmented with clocks, i.e. real-valued variables increasing uniformly, that are compared within guards and invariants with timing delays [3]. Parametric timed automata (PTA) [4] extend timed automata with parameters, i.e. unknown constants, that can be used in guards and invariants. We will use here an extension of PTA with *stopwatches* [2], where clocks can be stopped in some control states of the automaton.

Given a set X of clocks and a set U of parameters, a constraint C over X and U is a conjunction of linear inequalities on X and U[2]. Given a parameter valuation (or point) π, we write $\pi \models C$ when

[1] Unlike in [28], for simplicity in this paper we only consider upper bound arrival curves.

[2] Note that this is a more general form than the strict original definition of PTA [4]; since most problems for PTA are undecidable anyway, this has no practical incidence, and increases the expressiveness of the formalism.

the constraint where all parameters within C have been replaced by their value as in π is satisfied by a non-empty set of clock valuations.

Definition 1. *A parametric timed automaton with stopwatches (PSA) $\mathscr{A}$ is $(\Sigma, Q, q_0, X, U, K, I, slope, \rightarrow)$ with Σ a finite set of actions, Q a finite set of locations, $q_0 \in Q$ the initial location, X a set of h clocks, U a set of parameters, K a constraint over U, I the invariant assigning to every $q \in Q$ a constraint over X and U, $slope : Q \rightarrow \{0,1\}^h$ assigns a constant slope to every location, and $\rightarrow$ a step relation consisting of elements (q, g, a, ρ, q'), where $q, q' \in Q$, $a \in \Sigma$, $\rho \subseteq X$ is the set of clocks to be reset, and the guard g is a constraint over X and U.*

The *slope* function is the extension of parametric timed automata to *stopwatch* timed automata, since it allows one to stop the time elapsing of some clock variables in some locations. This expressive power is used in the context of schedulability to model the preemption mechanism.

It is well-known that the parallel composition (using a synchronisation on actions) of several PSA is itself a PSA. Hence, it is common to model a complex system by composing several system components modelled themselves using PSA.

The semantics of a PSA $\mathscr{A}$ is defined in terms of states, i.e. pairs (q, C) where $q \in Q$ and C is a constraint over X and U. Given a point π, we say that a state (q, C) is π-compatible if $\pi \models C$. Runs are alternating sequences of states and actions, and traces are time-abstract runs, i.e. alternating sequences of *locations* and actions. The trace set of $\mathscr{A}$ corresponds to the traces associated with all the runs of $\mathscr{A}$. Given $\mathscr{A}$ and π, we denote by $\mathscr{A}[\pi]$ the (non-parametric) timed stopwatch automaton where each occurrence of a parameter has been replaced by its constant value as in π. Details can be found in, e.g. [8].

3.3 The Inverse Method

The inverse method for PSA [8] exploits the knowledge of a reference point of timing values for which the good behaviour of the system is known. The method synthesises automatically a dense space of points around the reference point, for which the discrete behaviour of the system, that is the set of all the admissible sequences of interleaving events, is guaranteed to be the same.

The inverse method IM proceeds by exploring iteratively longer runs from the initial state. When a π-incompatible state is met (that is a state (q, C) such that $\pi \not\models C$), a π-incompatible inequality J is selected within the projection of C onto U. This inequality is then negated, and the analysis restarts with a model further constrained by $\neg J$. When a fixpoint is reached, that is when no π-incompatible state is found and all states have their successors within the set of reachable states, the intersection of all the constraints onto the parameters is returned.

IM proceeds by iterative state space exploration, and its result comes under the form of a fully parametric constraint. By repeatedly applying the method, we are able to decompose the parameter space into a covering set of "tiles", which ensure a uniform behaviour of the system: it is sufficient to test only one point of the tile in order to know whether or not the system behaves correctly on the whole tile. This is known as the *behavioural cartography* [5]. Both the inverse method and the behavioural cartography are semi-algorithms; that is, they are not guaranteed to terminate but, if they do, their result is correct.

4 A Modular Framework for Modelling Real-Time Systems

In this section we refer to a real-time system as a set of real-time tasks scheduled by a FPPS on a single processor. Of course, the discussion is valid also when considering a single component of a large real-time distributed system.

Our model of a real-time system consists of three kinds of PSA components: the task automata, the task activation automata and the scheduler automaton. We refer to the composition of these PSA components through synchronisation labels as the system automaton.

Each task is modelled using a task automaton. Such a task automaton is shown in Figure 2a. Each task automaton contains two (local) continuous clock variables c and d. Clock c counts the execution of the task and clock d counts the time passed since last job arrival. Since we consider generic activation patterns (periodic, sporadic or arrival curves), a new instance may be activated while the previous ones have not yet completed. Hence, there could be several active jobs from the same task at the same time. A discrete[3] variable N is used to count the number of simultaneous active instances for the task.

Initially, a task is in location Idle. The synchronisation label arrival_event notifies that a new instance from this task is activated and triggers a transition to a committed location ActEvent. A committed location is a location where time elapsing is not allowed, represented graphically using a double circle location. The label arrival is used between a task and the scheduler. The task will then go to location Waiting and wait there for the scheduler's decision whether to occupy the CPU. If a task has the highest priority among the active tasks in the system, the scheduler will send dispatch to trigger the transition from Waiting to Running. While a task is in Running, the scheduler could revoke the CPU for a higher priority task through synchronisation label preemption.

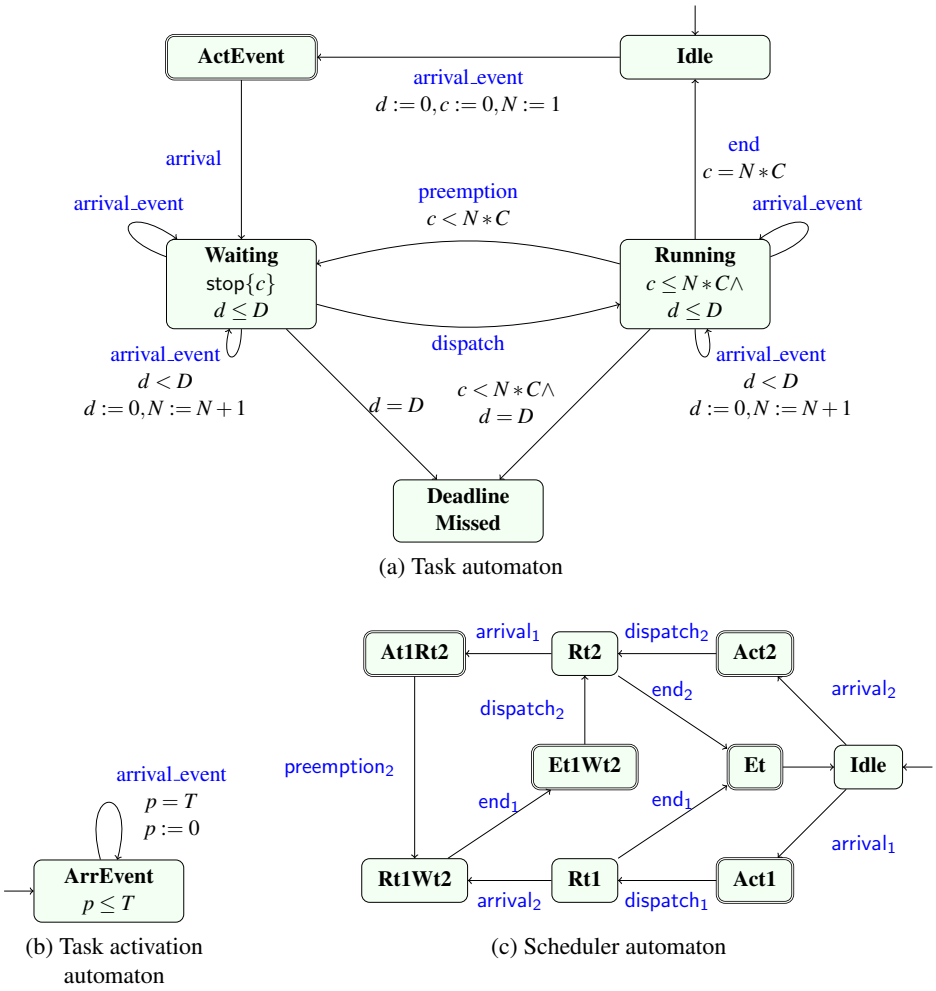
Clock d always progresses and the execution time clock variable c is stopped if a task is waiting. When a task is waiting for the CPU or running on the CPU, to react to new activations, it will non-deterministically choose to increase the counter N of active instances by 1. When a job misses its deadline ($d = D$) before completing its execution, it will go to DeadlineMissed. When a task finishes its execution ($c = N * C$), it will go back to initial location Idle.

There could be many different activation patterns for a task, such as periodic, sporadic or according to arrival curves. We only require that the activation automaton synchronises with the task automaton on label arrival_event. As a demonstration, Figure 2b shows the activation model for a periodic task. Every period T, the automaton sends the signal arrival_event to inform the arrival of a new job.

In this paper, we assume tasks are scheduled according to a fixed-priority fully-preemptive scheduler (FPPS). The scheduler automaton synchronises with the tasks and decides which task will occupy the CPU at each time. The structure of the automaton is completely fixed given a number of tasks.

Figure 2c shows a scheduler for two tasks, where task 1 has higher priority. The scheduler automaton can be expanded in a similar form to deal with a task set with more tasks. In the scheduler automaton, the labels arrival, dispatch, preemption and end are the same as in task automaton; we append a label with index i, e.g. end_i, to denote that this label synchronises with task i. The convention we use for naming the location encodes the status of the tasks: Rtx means the task τ_x is running; Atx means task τ_x is just activated; Wtx means task τ_x is waiting; Et is saying the task just finished its execution.

[3] Discrete variables are not part of the original PTA/PSA formalisms, but can seen as syntax sugar to increase the number of discrete states (locations). Such discrete variables are supported by most tools for (parametric) timed automata.



(a) Task automaton

(b) Task activation automaton

(c) Scheduler automaton

Figure 2: The modelling framework for a real-time system

5 Parametric Schedulability Analysis of Real-Time Components

5.1 Convergence Problem

We first show that the application of the inverse method IM to a system with parametric task activations does not yield satisfactory results. Consider a task set with two periodic tasks $\tau_1 = (31, T_1, T_1), \tau_2 = (49, T_2, T_2)$ with implicit deadlines (i.e. deadlines always equal to periods). If we use IM with initial values $T_1 = 60$ and $T_2 = 120$, respectively, the final constraints obtained will be $T_1 = 60$ and $T_2 = 120$. That is, the result produced by IM is a single point, the initial valuation.

Such result is caused by an important property of the schedule. The inverse method synthesises a set of constraints that delimit the values for the parameters that result in the same exact traces as the initial valuation. The schedule generated by a set of periodic real-time tasks is itself periodic with period H (also called *hyperperiod*). In particular, the sequence of scheduling events repeats itself every H, and different H will result in different traces of task execution. The hyperperiod can be computed as the least common multiple of all task periods: $H = \mathsf{lcm}(T_1, \ldots, T_n)$. When periods are parametric, and since

function lcm() is highly non linear, a small variation on one period can cause very large variations in the hyperperiod. For example, consider the two previous tasks with initial valuation of the periods $T_1 = 60$ and $T_2 = 120$, respectively. Their hyperperiod is 120. When we increase the second period to 121, the hyperperiod becomes 7260. Clearly, in this second case the traces are much longer and contain many more events. This explains why IM only converges to the initial valuation.

Of course, things become even more complex when considering generic arrival patterns. The next section solves this convergence problem by exploiting a well-known result from classical scheduling theory.

5.2 An Improved Model of the System

As discussed in Section 5.1, it is infeasible to apply IM directly to a system model with parametric arrival patterns. We will try to avoid this situation by adapting the system automaton (Figure 2) by exploiting the concept of critical instant.

For a set of periodic or sporadic tasks scheduled by FPPS on a single processor it is possible to define a *critical scenario*, which is the situation that arises when all tasks are simultaneously activated (*critical instant*) and every task τ_i generates subsequent jobs as soon as it is allowed. According to the seminal work by Liu and Layland [24], the worst-case response time of a task can be found in the *busy period* (i.e. interval in which the processor is continously busy) that starts at the critical instant.

This means that, if we want to check the schedulability of a set of periodic or sporadic real-time tasks, it is sufficient to activate all tasks at time zero and check that no deadline is missed in the first busy period starting at time 0. Therefore, as soon as the processor becomes idle we can stop our search.

In the system automaton in Section 4, each trace corresponds to a possible schedule of the task set. However, we now know that to check the schedulability of a task set, it is sufficient to analyse traces starting from the critical instant till the first idle time in CPU. So, we adapt the system automaton as follows:

- The task activation automaton is required to release its first job at time 0 and it will emit the subsequent jobs as fast as the task is allowed;
- In the scheduler automaton, after all tasks complete their execution, instead of going back to Idle, it will transit from Et to a new location Stop, where this is no outgoing edge.

The first point is used to simulate the worst-case behaviour of tasks at the critical instant. Rather than going to Idle and waiting for new task releases, the scheduler automaton (also the system automaton) simply stops. We call this adapted scheduling model as the *idle-time scheduler automaton*.

The idle-time scheduler automaton actually simulates the longest *busy period*, which starts from the critical instant and ends at the first idle time of the processor. The length of this busy period depends both on the execution time and on activation periods of the tasks. However, the dependence from the periods is not so strong as with the hyperperiod. Let us consider again the previous set of two periodic tasks $\tau_1 = (C_1 = 31, T_1 = D_1 = 60)$ $\tau_2 = (49, 120, 120)$. The schedule for the first busy period is shown in Figure 3. Task τ_1 executes twice before the first instance of τ_2 can complete.

The length of the busy period in this case is $2C_1 + C_2 = 111$. By doing some simple calculation, it is easy to see that changing T_1 to any value in $[56, 79]$ does not change the sequence of events in the busy period: in facts, for any value of T_1 in that interval, τ_1 will still execute two times before the first instance of τ_2 completes. Also, changing T_2 to any value $T_2 \geq 112$ does not change the busy period.

Hence, we can apply IM on the new model and avoid the convergence problem as in Section 5.1. Let us assume $T_1 \in [40, 120]$, $T_2 \in [80, 200]$ and let us apply the behavioural cartography to obtain the

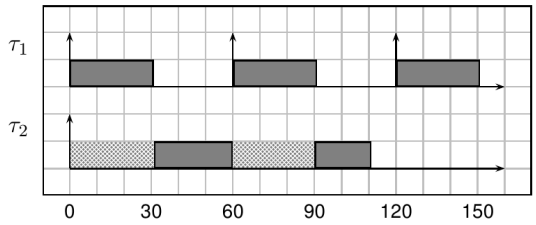


Figure 3: Schedule of the first busy period of the example task set

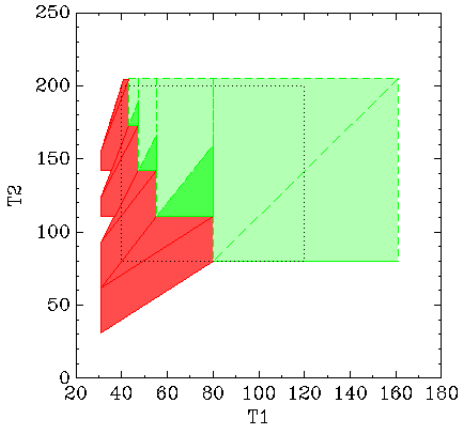


Figure 4: Constraints on T_1 and T_2 obtained by the behavioural cartography

constraint space of T_1, T_2 that keeps the task set schedulable. The result is given in Figure 4 in a graphical form. The red part (on the left) is the constraint space on T_1 and T_2 in which the system misses τ_2's deadlines, whereas the green part (on the right) is where no deadline is missed.

When applying the behavioural cartography to the idle-scheduler automaton, there may exist a combination of parameters that cause the system to go into overload, i.e. there will be no idle time in the schedule. For example, in case of periodic tasks, this happens when the total system utilisation is such that $\sum_{i=1}^{n} \frac{C_i}{T_i} > 1$. In the previous example, $(T_1 = 40, T_2 = 80)$ is one such point. Of course, this will surely cause a deadline miss, because it means that the total amount of work to be performed (utilisation) exceeds the amount of available processor time.

To solve this case, we put an upper bound on the maximal depth of the traces computed by IM. This bound is always computable in the case of periodic real-time tasks, and corresponds to computing an upper bound to the time where a deadline miss will happen. A method for computing such a bound can be built by using the concept of *demand bound function* [9].

5.3 Applicability of the Idle-Time Scheduler

It is possible to prove that the concepts of critical instant and maximal busy periods are valid also when considering tasks activated by generic arrival curves [28]. In particular, the critical scenario corresponds to the time instant in which all tasks are activated with their initial burstiness (critical instant), and their successive instances arrive as soon as possible without violating their arrival curves. Then, the worst-case response time can be found in the busy period starting at the critical instant and corresponding to

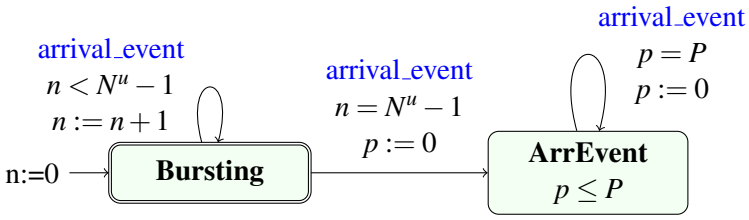


Figure 5: Arrival curve automaton

the critical scenario. Therefore, we will use the same technique also for generic arrival curves.

In Figure 5 we show the simple PSA model for a parametric periodic arrival curve described by Equation 1. Initially, the arrival curve automaton is in a committed location Bursting with $n = 0$, where n is a discrete variable counting the number of initial client requests. The automaton emits N^u activations for a task (τ_2 in our case) within 0 time elapse and then moves to location ArrEvent where is starts behaving as a periodic activation automaton as in Figure 2b, and produces activations events every P.

For other different task models there is no critical instant. For example, when considering periodic tasks with initial offset different from zero, there is no worst-case scenario in the schedule. Instead, it is necessary to analyse all busy periods in the interval $[0, 2H + \Phi_{\mathsf{max}}]$, where Φ_{max} is the largest initial offset [21].

Given a task set $\mathscr{T}$ of periodic real-time task with offsets, we can build a task set $\mathscr{T}'$ that contains the same tasks with the same parameters except that their initial offsets are all set to zero. In this case, it is possible to prove that, if $\mathscr{T}'$ is schedulable, then also $\mathscr{T}$ is schedulable. However, the converse does not hold. Therefore, it is possible to perform a parametric analysis of $\mathscr{T}'$ using our idle-time scheduler, and the set of values of the parameters produced by the analysis is a subset of the set of valid parameters for the original system $\mathscr{T}$. A more precise analysis requires point-by- point exploration of the parameter space.

Finally, in this paper we assume that task are independent from each other, and do not self-suspend waiting for other events different from the activation event. An example of self-suspending task is a task that performs a remote procedure call, and self-suspends waiting for the response. Again, in this case there is not a single critical scenario for the task set, therefore our simplified model cannot be used.

6 Towards Timed Interfaces

In this section we show how it is possible to define a *timed interface* of a real-time component using parametric analysis.

Consider the system of Figure 1: it consists of 3 tasks τ_1, τ_2 and τ_3 running on a single processor with FPPS. A task with smaller index has higher priority. τ_1 and τ_3 are periodic tasks with $\tau_1 = (C_1 = 2, D_1 = 8, T_1 = 8)$ and $\tau_3 = (C_3 = 20, D_3 = 50, T_3 = 50)$. Task τ_2 has $C_2 = 5$ and implements the method provided in the interface. We assume that this component is linked to a local networks, and task τ_2 receives the requests from clients running on other nodes of the network. We would like to know how many clients can ask requests to the system, with which frequency, and the maximum delay that is going to pass from the request to the response. Therefore, we need to study the possible activation patterns of task τ_2 and its worst-case response time. For modelling the activation patterns, we use a parametric arrival curse as described by Equation 1. For example, $N^u = 2$ and $P = 100$ means that we can connect at most 2 independent clients, and that between any two consecutive requests after the first two there must be at

$N^u = 1$	when $(20 \leq P \leq 26) \vee (27 \leq P \leq 34) \vee (35 \leq P \leq 50) \rightarrow D_2^{min} = 10$
$N^u = 2$	when $(24 \leq P \leq 26) \vee (27 \leq P \leq 34) \vee (35 \leq P \leq 50) \rightarrow D_2^{min} = 14$
$N^u = 3$	when $(P = 47) \vee (48 \leq P \leq 50) \rightarrow D_2^{min} = 21$

Table 1: The final interface

most 100 units of time.

Both N^u and P are parameters we are going to synthesise with our parametric analysis. Another parameter is the delay (deadline) D_2 of τ_2. We are interested in the parameter space that guarantees all the tasks are schedulable.

First, we construct the activation automaton for $\alpha(t)$ as in Figure 5. Following the method described in Section 4, and using the idle-time scheduler automaton, we then compose the final automaton.

Given that $C_2 = 5$, it is easy to see that the burst (N^u) of the arrival curve automaton cannot be larger than 3, otherwise τ_3 will be doomed to miss its deadline, because $D_3 < C_3 + 4C_2 + 5C_1$. Additionally, we assume P and D_2 lie in following intervals:

$$P \in [20,50],\ D_2 \in [10,50]$$

N^u is a discrete parameter that must be treated separately from the other parameter. Our strategy is to instantiate N^u with 1, 2 and 3 individually and apply IM to each case in order to synthesise constraints over P and D_2 that keep the system schedulable. The resulted parameter spaces for the three cases are visualised in Figure 6.

We can use these values to build a *timed interface specification* for the component.

- the number of distinct independent clients that can be connected to the service must respect the constraint $1 \leq N^u \leq 3$;
- Depending on the number of clients, the relationship between minimum request period P and worst case response time D_2 is specified in Table 1.

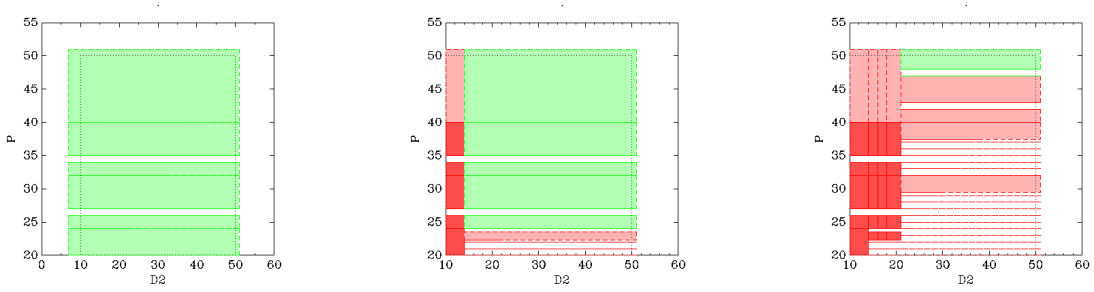
Reducing the number of regions

As it is possible to see in Figure 6 and in Table 1, the parameter space returned by IM consists of a set of disjoint tiles. Each tile is a convex region and the resulting interface is the union of (maybe a large number of) these convex regions. Such an interface may not be easy to use due to the large number of disjoint regions.

In some cases, it is possible to perform a "merge" operation between the tiles, as explained in [6], in order to reduce the number of convex regions composing the final interface. Two convex regions are mergeable if their convex hull equals to their union. Given tiles returned from IM, we repeatedly replace mergeable tiles, satisfying this condition, with their union till there are no mergeable tiles. If we restrict ourselves to integers solutions, we may further merge adjacent tiles. For example, the constraint $(20 \leq P \leq 26)$ can be merged with $(27 \leq P \leq 34)$, thus obtaining $(20 \leq P \leq 34)$. We are currently investigating efficient methods for automatically merging tiles resulting from IM cartography.

7 Conclusion and Future Work

In this paper we have presented a PTA model of a real-time systems scheduled by FPPS. We have shown how to perform a parametric analysis using IM with a specific model of the scheduler that stops at the



(a) $N^u = 1$ (b) $N^u = 2$ (c) $N^u = 3$

Figure 6: Parameter space (green) for N^u, P and D_2

first idle time. Finally, we have shown how to use parametric analysis for the design and the specification of the interface of a real-time component.

We wish to continue along this line of research and investigate about the possibility to systematically use parametric analysis for interface specification. We are currently investigating efficient methods for reducing the complexity of the set of regions produced by IM, either by using more sophisticated merging techniques, or by using conservative approximations. Also, we plan to extend the analysis to more complex task models like self-suspending tasks and task dependencies.

More specifically on the parameter synthesis techniques, it would be interesting to reuse some technique for integer parameter synthesis recently proposed in [18]; on the negative side, only integer points are synthesised, thus preventing the interpretation of the result for robustness analysis (in the sense of infinitesimal variations of the parameters); on the positive side, these techniques are efficient and guaranteed to terminate. Also, combining the inverse method with IC3 [13] is an interesting future direction of research.

A more general (and challenging) objective is also to be able to derive (possibly non-linear) constraints relating the discrete and continuous parameters, e.g. relating the number of clients ("N^u" in Section 6) with the timing parameters ("P" and "D_2" in Section 6).

Acknowledgment

We would like to thank anonymous reviewers for their useful comments.

References

[1] (1999): *IEEE Standard for Information Technology-Portable Operating System Interface (POSIX)-Part 1: System Application Program Interface (API)- Amendment D: Additional Real time Extensions [C Language]*, doi:10.1109/IEEESTD.1999.91515.

[2] Yasmina Adbeddaïm & Oded Maler (2002): *Preemptive Job-Shop Scheduling using Stopwatch Automata*. In: *TACAS*, *Lecture Notes in Computer Science* 2280, Springer-Verlag, pp. 113–126, doi:10.1007/3-540-46002-0_9.

[3] Rajeev Alur & David L. Dill (1994): *A theory of timed automata*. Theoretical Computer Science 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[4] Rajeev Alur, Thomas A. Henzinger & Moshe Y. Vardi (1993): *Parametric real-time reasoning*. In: *STOC*, ACM, pp. 592–601, doi:10.1145/167088.167242.

[5] Étienne André & Laurent Fribourg (2010): *Behavioral Cartography of Timed Automata*. In: *RP*, Lecture Notes in Computer Science 6227, Springer, pp. 76–90, doi:10.1007/978-3-642-15349-5_5.

[6] Étienne André, Laurent Fribourg & Romain Soulat (2013): *Merge and Conquer: State Merging in Parametric Timed Automata*. In: *ATVA*, Lecture Notes in Computer Science 8172, Springer, pp. 381–396, doi:10.1007/978-3-319-02444-8_27.

[7] Étienne André, Laurent Fribourg, Ulrich Kühne & Romain Soulat (2012): *IMITATOR 2.5: A Tool for Analyzing Robustness in Scheduling Problems*. In: *FM*, Lecture Notes in Computer Science 7436, Springer, p. 33–36, doi:10.1007/978-3-642-32759-9_6.

[8] Étienne André & Romain Soulat (2013): *The Inverse Method*. ISTE Ltd and John Wiley & Sons Inc., doi:10.1002/9781118569351.

[9] Sanjoy K. Baruah, Louis E. Rosier & Rodney R. Howell (1990): *Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor*. Real-Time Systems 2(4), pp. 301–324, doi:10.1007/BF01995675.

[10] Enrico Bini, Marco Di Natale & Giorgio C. Buttazzo (2006): *Sensitivity Analysis for Fixed-Priority Real-Time Systems*. In: *ECRTS*, p. 13–22, doi:10.1007/s11241-006-9010-1.

[11] Giacomo Bucci, Andrea Fedeli, Luigi Sassoli & Enrico Vicario (2004): *Timed state space analysis of real-time preemptive systems*. IEEE Transactions on Software Engineering 30(2), pp. 97–111, doi:10.1109/TSE.2004.1265815.

[12] Giorgio C. Buttazzo (2004): *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag TELOS.

[13] Alessandro Cimatti, Alberto Griggio, Sergio Mover & Stefano Tonetta (2013): *Parameter synthesis with IC3*. In: *FMCAD*, IEEE, pp. 165–168, doi:10.1109/FMCAD.2013.6679406.

[14] Alessandro Cimatti, Luigi Palopoli & Yusi Ramadian (2008): *Symbolic Computation of Schedulability Regions Using Parametric Timed Automata*. In: *RTSS*, IEEE Computer Society, pp. 80–89, doi:10.1109/RTSS.2008.36.

[15] Elena Fersman, Leonid Mokrushin, Paul Pettersson & Wang Yi (2003): *Schedulability Analysis Using Two Clocks*. In: *TACAS*, pp. 224–239, doi:10.1007/3-540-36577-X_16.

[16] Elena Fersman, Paul Pettersson & Wang Yi (2002): *Timed Automata with Asynchronous Processes: Schedulability and Decidability*. In: *TACAS*, Springer-Verlag, pp. 67–82, doi:10.1007/3-540-46002-0_6.

[17] Gomaa Hassan (2012): *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*. Cambridge University Press, doi:10.1145/1988997.1989008.

[18] Aleksandra Jovanović, Didier Lime & Olivier H. Roux (2013): *Integer Parameter Synthesis for Timed Automata*. In: *TACAS*, Lecture Notes in Computer Science 7795, Springer, pp. 401–415, doi:10.1007/978-3-642-36742-7_28.

[19] Kai Lampka, Simon Perathoner & Lothar Thiele (2013): *Component-based system design: analytic real-time interfaces for state-based component implementations*. International Journal on Software Tools for Technology Transfer 15(3), p. 155–170, doi:10.1007/s10009-012-0257-7.

[20] Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone & Yusi Ramadian (2013): *Timed-automata based schedulability analysis for distributed firm real-time systems: A case study*. International Journal on Software Tools for Technology Transfer 15(3), pp. 211–228, doi:10.1007/s10009-012-0245-y.

[21] Joseph Y.-T. Leung & Jennifer Whitehead (1982): *On the complexity of fixed-priority scheduling of periodic, real-time tasks*. Performance Evaluation 2(4), pp. 237–250, doi:10.1016/0166-5316(82)90024-4.

[22] Didier Lime & Olivier H. Roux (2009): *Formal verification of real-time systems with preemptive scheduling*. *Real-Time Systems* 41(2), pp. 118–151, doi:10.1007/s11241-008-9059-0.

[23] Shang-Wei Lin, Étienne André, Yang Liu, Jun Sun & Jin Song Dong (2014): *Learning Assumptions for Compositional Verification of Timed Systems*. *Transactions on Software Engineering*, doi:10.1109/TSE.2013.57. To appear.

[24] Chung Laung Liu & James W Layland (1973): *Scheduling algorithms for multiprogramming in a hard-real-time environment*. *Journal of the ACM* 20(1), pp. 46–61, doi:10.1145/321738.321743.

[25] Jane W. Liu (2000): *Real-time systems*. Prentice Hall PTR.

[26] Insik Shin & Insup Lee (2008): *Compositional real-time scheduling framework with periodic model*. *ACM Transactions on Embedded Computing Systems (TECS)* 7(3), p. 30, doi:10.1145/1347375.1347383.

[27] Youcheng Sun, Romain Soulat, Étienne Lipari, Giuseppe André & Laurent Fribourg (2013): *Parametric Schedulability Analysis of Fixed Priority Real-Time Distributed Systems*. In Cyrille Artho & Peter Ölveczky, editors: *Second International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'13)*, *Communications in Computer and Information Science* 419, Springer. To appear.

[28] L. Thiele, S. Chakraborty & M. Naedele (2000): *Real-time calculus for scheduling hard real-time systems*. In: *Proceedings of the 2000 IEEE International Symposium on Circuits and Systems, ISCAS 2000, Geneva.*, 4, IEEE, pp. 101–104, doi:10.1109/ISCAS.2000.858698.

[29] Louis-Marie Traonouez, Didier Lime & Olivier H. Roux (2009): *Parametric Model-Checking of Stopwatch Petri Nets*. *Journal of Universal Computer Science* 15(17), pp. 3273–3304, doi:10.3217/jucs-015-17-3273.

Worst-case Throughput Analysis for Parametric Rate and Parametric Actor Execution Time Scenario-Aware Dataflow Graphs

Mladen Skelin
Norwegian University of Science and Technology
Trondheim, Norway

Marc Geilen
Eindhoven University of Technology
Eindhoven, The Netherlands

Francky Catthoor
IMEC vzw.
Leuven, Belgium

Sverre Hendseth
Norwegian University of Science and Technology
Trondheim, Norway

Scenario-aware dataflow (SADF) is a prominent tool for modeling and analysis of dynamic embedded dataflow applications. In SADF the application is represented as a finite collection of synchronous dataflow (SDF) graphs, each of which represents one possible application behaviour or scenario. A finite state machine (FSM) specifies the possible orders of scenario occurrences. The SADF model renders the tightest possible performance guarantees, but is limited by its finiteness. This means that from a practical point of view, it can only handle dynamic dataflow applications that are characterized by a reasonably sized set of possible behaviours or scenarios. In this paper we remove this limitation for a class of SADF graphs by means of SADF model parametrization in terms of graph port rates and actor execution times. First, we formally define the semantics of the model relevant for throughput analysis based on (max,+) linear system theory and (max,+) automata. Second, by generalizing some of the existing results, we give the algorithms for worst-case throughput analysis of parametric rate and parametric actor execution time acyclic SADF graphs with a fully connected, possibly infinite state transition system. Third, we demonstrate our approach on a few realistic applications from digital signal processing (DSP) domain mapped onto an embedded multi-processor architecture.

1 Introduction

Synchronous dataflow (SDF) [19] was introduced as a restriction of Kahn process networks (KPN) [18] to allow compile-time scheduling. The term *synchronous* means *static* or *regular*. Synchronous dataflow graphs (SDFGs) are directed graphs where nodes are called *actors* and edges are called *channels*. The numbers of data samples produced or consumed are known at compile time. We refer to these data samples as *tokens* and to the token production and consumption numbers as *rates*. Although SDF is very fitted to model regular streaming applications, it is due to its static nature, very lacking in its ability to capture the dynamic behaviour of modern streaming applications. Therefore, a notable number of SDF extensions has been proposed over the years. Cyclo-static dataflow (CSDF) [6] allows token production and consumption to vary between actor firings as long as the variation forms a certain type of a periodic pattern, while models such as parametrized synchronous dataflow (PSDF) [5], variable-rate dataflow (VRDF) [23], variable-rate phased dataflow (VPDF) [23] and schedulable parametric dataflow (SPDF) [10] introduce parametric rates. Scenario-aware dataflow (SADF) [22] encodes the dynamism of an application by identifying a finite number of different behaviours called *modes* or *scenarios*. Each of the modes is represented by a single synchronous dataflow graph. The modes or scenarios can occur in known or unknown sequences. A finite state machine (FSM) is used to encode occurrence patterns.

É. André and G. Frehse (Eds.): 1st International Workshop
on Synthesis of Continuous Parameters (SynCoP'14).
EPTCS 145, 2014, pp. 65–79, doi:10.4204/EPTCS.145.7

SADF is equiped with a technique that yields the tightest possible performance guarantees [12]. The power of this technique lies in its ability to consider transitions over all possible scenario sequences as given by the FSM. Considering only the worst-case scenario, i.e. the scenario with the lowest throughput, without considering scenario transitions could be too optimistic. On the other hand, merging all application SDFGs into one SDFG where an actor takes the worst-case execution time over all SDFGs in SADF would be too pessimistic. This is due to the fact that subsequent iterations belonging to different scenarios may overlap in time, i.e. execute in a pipelined fashion. However, SADF is limited by its finiteness. It can only handle a reasonably sized set of application scenarios.

To illustrate this, let us define an abstract parallel application consisting of a nested *for* loop with parametric affine loop bounds:

```
ProcessData.A(out g, out h);

for (i=0; i<=g; i++){
   for (j=0; j<=h; j++){
      // Perform two tasks in parallel
      #region ParallelTasks
      // Perform two tasks in parallel
      Parallel.Invoke(() =>
         {
            ProcessData.B(i,j);
         }, // close first parallel action
         () =>
         {
            ProcessData.C(i,j);
         } // close second parallel action
      ); // close Parallel.Invoke
      #endregion

      ProcessData.D(i,j);
   }
}
```

The example application consists of 4 subtasks: `ProcessData.A`, `ProcessData.B`, `ProcessData.C` and `ProcessData.D` with known worst-case execution times. Data parallelism is elegantly specified using the `Parallel.Invoke` construct. Inside the `Parallel.Invoke` construct, an *Action* delegate is passed for each item of work. The application is mapped onto a multi-processor platform. The task assignment employed is purely static. In order to add complexity, we assume that the application executes in a pipelined fashion, i.e. more instances of the application can be active at the same time. Such an assumption introduces resource dependencies over subsequent activations of the application. In other words, a subtask of the $(i+1)^{\text{th}}$ activation of the application might have to wait for a certain subtask of the i^{th} activation to complete and release the corresponding processing element. As specified by the example code, g and h can take different values during each application execution, i.e. they are data-dependant and are the result of input data processing performed by the subtask `ProcessData.A`. Let us assume we know that g can take the value from the interval $\left[0, \frac{n}{2}\right]$ and h can take the value from the interval $\left[0, \frac{m}{2}\right]$. In that case, from a pure timing perspective, this application will exhibit as many behaviours as there are integer points in the rational 2-polytope $P_{n,m}$ given by the set of constraints

$\{0 \leq \frac{1}{2}n,\ 0 \leq \frac{1}{2}m\}$. For $n = 4500$ and $m = 2001$, to be able to use SADF to derive the tightest worst-case performance bounds, even for such a simple application executing in a pipelined fashion on a multi-processor platform, we would have to generate $2,252,126$ SDFGs [8]. The situation gets even worse when dealing with platforms that support dynamic voltage and frequency scaling (*DVFS*), which is a commonly used technique that adapts both voltage and frequency of the system in respect to changing workloads [20]. In this case also the execution times of the application subtasks would vary depending on the current DVFS setting of the processing element they are mapped to.

In our work, we will remove these limitations which hamper the use of SADF in important application domains. For this purpose, we will add parametrization to the basic SADF modeling approach both in terms of parametric rates and parametric actor execution times given over a parameter space, which is a totally non-trivial extension because the current core of the SADF framework relies strongly on the constant nature of the rates and actor execution times. We raise the problem of SADF parametrization in the scope of existing parametric dataflow models. PSDF [5] and SPDF [10] are two semantically very similar models that provide a high level of generalization. We prefer SPDF due to syntactical convenience. By incorporating SPDF semantics into the definition of our parametric rate and parametric actor execution time SADF (PSADF), we show that the SPDF model can at run-time be treated as a special case of a SADF. We then derive a technique for worst-case throughput analysis for PSADF. We demonstrate our approach on a few realistic applications from the digital signal processing (DSP) domain.

2 Related Work

Throughput analysis of SDFGs is studied by many authors. Reference [15] gives a good overview of the existing methods. Due to the static nature of SDF, these methods cannot be applied to any form of parametric dataflow. [14] presents three methods for throughput computation for an SDFG where actor execution times can be parameters. However, the technique does not consider parametric rates and can only handle the static case, i.e. the graph cannot change parameter values during its execution. [12] introduces the (max,+) semantics for the SADF model relevant for worst-case performance analysis, but is, as previously mentioned, practically limited to a reasonably sized set of scenarios. The most closely related work to ours can be found in [9]. It combines the approaches presented in [12] and [14] and yields a technique that finds throughput expressions for an SADFG where actors can have parameters as their execution times. However, the (max,+) semantics introduced in [9] can consider only parametric actor execution times and not parametric rates. A straightforward extension of [9] to cover the case of parametric rates is not possible because it is not clear how to symbolically execute the graph in the presence of parametric rates. In the scope of rate parametric dataflow models [5][10], little attention has been given to the aspect of time. Two examples of parametric models that explicitly deal with time are VRDF [23] and VPDF [23]. These address the problem of buffer capacity computation under a throughput constraint, but both have a structural constraint that each production of p tokens must be matched by exactly one consumption of p tokens. That drastically limits the scope of applications it can consider.

So, the current approaches in throughput analysis for dataflow MoCs either cannot consider parametric rates [15][14][12][9], or impose too hard structural constraints that severely limit the expressivity of the model [23]. In our work we will remove these limitations by embedding the SPDF model [10] which provides a high level of generalization into the SADF model [22][12].

3 Preliminaries

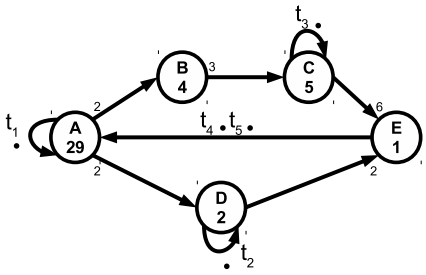
3.1 Synchronous Dataflow Graphs

SDFG is a directed graph $(\mathscr{A},\mathscr{E})$ where nodes represent *actors* which in turn represent functions or tasks, while edges represent their dependencies. We also refer to edges as *channels*. Execution of an actor is denoted as firing and it is assigned with a time duration. In SDF, the number of tokens consumed and produced by an actor is constant for each firing. We refer to these numbers as *rates*. Actors communicate using tokens sent over channels from one actor to another. Fig. 1a shows an example of an SDFG with 5 actors ($\mathscr{A} = \{A,B,C,D,E\}$) and 9 channels ($\mathscr{E} = \{(A,B),(B,C),(C,C),\ldots\}$). Some channels might contain initial tokens, depicted with solid dots. The example graph contains 5 initial tokens that are labeled $t_1,\ldots,t_5$. Each actor is assigned with a firing time duration, denoted in the actor node, below the actor name, e.g. actor A has a firing duration of 29 time-units. Each port is assigned with a rate. When the value is omitted, it means that the value equals to 1. As rates in SDF are constant for each firing, it is possible to construct a finite schedule (if it exists) that can be periodically repeated [19]. Such a schedule assures liveness and boundedness [19]. We call such minimal sequence of firings an iteration of the SDFG. This is a sequence of firings that has no net effect on the token distribution in the graph. The numbers of firings of each actor within an iteration constitute the *repetition vector* of an SDFG. We only consider dataflow graphs that are bounded and live. Throughput is considered in terms of the number of iterations per time-unit, i.e. the number of iterations executed in one period normalized by the repetition vector divided by the duration of the period [15]. It is natural to do so, because an iteration represents a coherent set of calculations, e.g. decoding of a video frame. For more details we refer to [19][15].

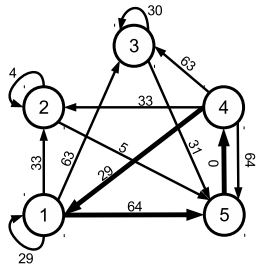
3.2 (max,+) Algebra for SDFGs

Let $a \oplus b = max(a,b)$, $a \otimes b = a+b$ for $a,b \in \overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$. By max-algebra we understand the analogue of linear algebra developed for the pair of operations $(\oplus,\otimes)$ extended to matrices and vectors [4]. Let $\vec{\gamma}$ denote the vector of production times of tokens that exist in their different channels in between iterations, i.e. it has an entry for each initial token in the graph. Then $\vec{\gamma}_k$ denotes the vector of production times of initial tokens after k iterations of the graph. These vectors then can be found using (max,+) algebra [4]. The evolution of the graph is then given by the following equation: $\vec{\gamma}_{k+1} = \mathbf{G}\vec{\gamma}_k$, where $\mathbf{G} = \{g_{ij}\}$ is a (max,+) characteristic matrix of the graph. Entry g_{ij} specifies the minimal elapsed time from the production time of the j^{th} token in the previous iteration to the production time of the i^{th} token in the current iteration. When the i^{th} token is not dependent on the j^{th} token, then $g_{ij} = -\infty$. The specification of the algorithm for obtaining $\mathbf{G}$ can be found in [13]. The (max,+) characteristic matrix for the example SDFG in Fig. 1a takes the form:

$$\mathbf{G} = \begin{bmatrix} 29 & -\infty & -\infty & 29 & -\infty \\ 33 & 4 & -\infty & 33 & -\infty \\ 63 & -\infty & 30 & 63 & -\infty \\ -\infty & -\infty & -\infty & -\infty & 0 \\ 64 & 5 & 31 & 64 & -\infty \end{bmatrix}.$$



(a) An example SDFG.



(b) MPAG of the example SDFG.

Figure 1: Synchronous dataflow

For example, $\vec{\gamma}_1$ can be calculated as below:

$$\vec{\gamma}_1 = \begin{bmatrix} 29 & -\infty & -\infty & 29 & -\infty \\ 33 & 4 & -\infty & 33 & -\infty \\ 63 & -\infty & 30 & 63 & -\infty \\ -\infty & -\infty & -\infty & -\infty & 0 \\ 64 & 5 & 31 & 64 & -\infty \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} max(29+0, 29+0) \\ max(33+0, 4+0, 33+0) \\ max(63+0, 30+0, 63+0) \\ max(0+0) \\ max(64+0, 5+0, 31+0, 64+0) \end{bmatrix} = \begin{bmatrix} 29 \\ 33 \\ 63 \\ 0 \\ 64 \end{bmatrix}.$$

Paper [12] explains how to obtain the throughput of an SDFG from the matrix **G**. Briefly, matrix **G** defines a corresponding (max,+) automaton graph (MPAG) [11]. MPAG has as many nodes as there are initial tokens in the graph. An edge with the weight g_{ij} is created from the j^{th} node to the i^{th} if $g_{ij} \neq -\infty$. The maximum cycle mean (MCM) λ of the MPAG identifies the critical cycle of the SDFG. The critical cycle limits the throughput of the SDFG which takes the value $1/\lambda$. MPAG of the example SDFG graph is displayed in Fig. 1b. The cycle with weights $g_{14} - g_{51} - g_{45}$ (denoted with bold arrows) determines the throughput which takes the value of $1/31$ iterations per time-unit.

3.3 Scenario-Aware Dataflow Graphs (SADFG)

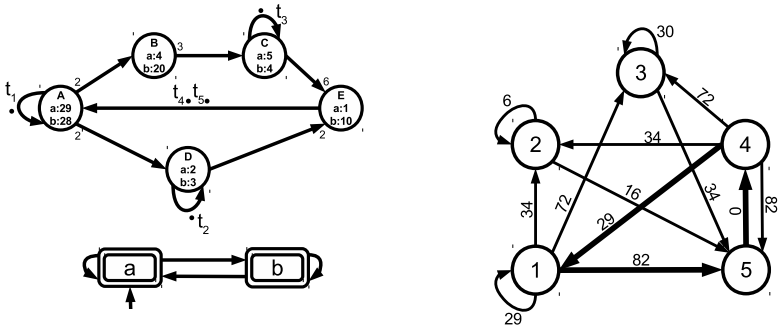
SADF models the dynamism of an application in terms of modes or scenarios. Every scenario is modeled by an SDFG, while the occurence patterns of scenarios are given by an FSM. We give the following definition of an SADFG.

Definition 1. *A Scenario-aware dataflow graph (SADFG) is a tuple* $SADFG = (S, F)$*, where:*

- $S = \{s_i \mid s_i = (scen_i, G_i)\}$ *is a set of ordered pairs of scenarios and their corresponding SDFGs;*
- $F = (Q, q_0, \delta, \Sigma, E)$ *is the scenario finite state machine consisting of a finite set Q of states, an initial state $q_0 \in Q$, a transition relation $\delta \subseteq Q \times Q$, a scenario labelling $\Sigma : Q \to S$ and a set of final states E, where $E = Q$.*

Fig. 2a shows an example SADFG with two scenarios, *a* and *b*. In this example both scenarios use the same scenario graph, but the actor execution times differ. For example, actor *A* has a firing duration of 29 time-units in scenario *a* and 28 time units in scenario *b*. The scenario FSM is fully connected and thus allowing arbitrary scenario order.

Every finite path of arbitrary length $\overline{q}$ over the FSM corresponds to a sequence $\overline{s}$ with $\overline{s}(k) = \Sigma(\overline{q}(k))$. When the FSM performs a transition, the SDFG graph associated with the destination state is executed for



(a) An example SADFG.

(b) MPAG of the examle SADFG.

Figure 2: Scenario-aware dataflow

exactly one iteration. Let $\mathbf{G}(s_i)$ denote the $n \times n$ (max,+) characteristic matrix for the scenario s_i, where n is the number of initial tokens in the SADFG. Then the completion time of a k-long sequence of scenarios can then be defined as a sequence of (max,+) matrix multiplications $\mathscr{A}(s_1 \dots s_k) = \mathbf{G}(s_k) \dots \mathbf{G}(s_1)\vec{i}$, where $\vec{i}$ specifies the initial enabling times of the graph's initial tokens and usually $\vec{i} = \vec{0}$. The worst case increase of $\mathscr{A}(\overline{s})$ for a growing length of $\overline{s}$ specifies the worst-case throughput for any sequence of scenarios [11] [12]. Reference [12] explains how to build the MPAG of an SADFG. Again, the inverse of the MCM $(1/\lambda)$ of the obtained MPAG denotes the worst-case throughput of that particular SADFG. A special case that arises in practice, which will be of the utmost importance in our SADF parametrization, is when scenarios can occur in arbitrary order, yielding the SADF FSM to be fully connected and with a single state for each scenario. In that case, the throughput of an SADFG equals to the maximum cycle mean of the MPAG that corresponds to the (max,+) matrix $\mathbf{G} = \max_{q \in Q}(\mathbf{G}(\Sigma(q)))$ [12]. The operator *max* denotes taking the maximum of the elements of the individual scenario matrices. The corresponding scenario matrices for the example SADFG in Fig. 2a are:

$$\mathbf{G}(a) = \begin{bmatrix} 29 & -\infty & -\infty & 29 & -\infty \\ 33 & 4 & -\infty & 33 & -\infty \\ 63 & -\infty & 30 & 63 & -\infty \\ -\infty & -\infty & -\infty & -\infty & 0 \\ 64 & 5 & 31 & 64 & -\infty \end{bmatrix} \quad \mathbf{G}(b) = \begin{bmatrix} 28 & -\infty & -\infty & 28 & -\infty \\ 34 & 6 & -\infty & 34 & -\infty \\ 72 & -\infty & 24 & 72 & -\infty \\ -\infty & -\infty & -\infty & -\infty & 0 \\ 82 & 16 & 34 & 82 & -\infty \end{bmatrix}.$$

The critical cycle of the corresponding MPAG obtained from the maximized matrix $\mathbf{G} = max(\mathbf{G}(a), \mathbf{G}(b))$, is denoted by bold arrows in Fig. 2b. Throughput in this case equals $1/37$ iterations per time-unit. This example also demonstrates that the worst-case throughput value cannot simply be obtained by only considering the 'worst-case' scenario, or by analysing the graph where each actor takes its worst-case execution time over all scenarios.

4 Parametric Rate and Actor Execution Time SADF Analysis

We start this section by formally defining the PSADF model and showing the (max,+) equivalence between SADF and PSADF. We use this result in defining the PSADF worst-case throughput calculation problem as a constrained optimization problem over the PSADF graph (PSADFG) parameter space,

where the objective functions are elements of the symbolic PSADFG (max,+) characteristic matrix. We conclude by giving the theoretical foundation and the algorithm for symbolic PSADFG (max,+) characteristic matrix extraction.

4.1 Motivation and Model Definition

SADF becomes impractical or even infeasible when it faces applications with a vast set of possible behaviours. We overcome this limitation by *parametrization*. The problem of parametrization of a dataflow model in terms of rates is not an easy task as it raises questions about properties like liveness, boundedness and schedulability. A naive approach in just declaring any rate of interest as parametric, could render the graph to deadlock, be unbounded or unschedulable. Therefore we start from SPDF [10]. The liveness and boundedness properties for SPDF are decidable. SPDF extends SDF by allowing rates to be parametric while preserving static schedulability. Rates are products of static natural numbers and/or parameters that can change dynamically. The changes of each parameter p are made by a single actor called its modifier each α^{th} time it fires using '*set* $p[\alpha]$' annotation. We re-define SPDF [10] by adding the notion of time of SDF/SADF to it.

Definition 2. *A schedulable parametric dataflow graph (SPDFG) is a tuple* $SPDFG = (\mathscr{G}, \mathscr{PR}, \mathscr{PD}, i, r, e, M, \alpha)$*, where:*

- $\mathscr{G}$ *is a directed connected graph* $(\mathscr{A}, \mathscr{E})$ *with* $\mathscr{A}$ *set of actors and* $\mathscr{E} \subseteq \mathscr{A} \times \mathscr{A}$ *set of edges (channels);*
- $\mathscr{PR}$ *is a set of rate parameters (symbolic variables) used to define SPDF rates by the grammar* $\mathscr{FR} ::= k \mid pr \mid \mathscr{FR}_1 \cdot \mathscr{FR}_2$*, where* $pr \in \mathscr{PR}$*,* $k \in \mathbb{N}^+$*;*
- $\mathscr{PD}$ *is a set of actor execution time parameters (symbolic variables) used to define SPDF actor execution times by the grammar* $\mathscr{FD} ::= k \cdot pd \mid \mathscr{FD}_1 + \mathscr{FD}_2$*, where* $pd \in \mathscr{PD}$*,* $k \in \mathbb{R}_0^+$*;*
- $i : \mathscr{E} \to \mathbb{N}_0$ *returns for each edge channel its number of initial tokens;*
- $r : \mathscr{A} \times \mathscr{E} \to \mathscr{FR}$ *returns for each port (represented by an actor and one of its edges) its rate;*
- $e : \mathscr{A} \to \mathscr{FD}$ *returns for each actor its execution time;*
- $M : \mathscr{PR} \to \mathscr{A}$ *and* $\alpha : \mathscr{PR} \to \mathscr{FR}$ *returns for each rate parameter its modifier and its change period.*

We consider only live SPDFGs as defined in [10]. We allow parameters (rates and actor execution times) to change in between iterations. The introduction of parametric actor execution times to SPDF does not influence the liveness property. We define actor execution times as linear combinations of parameters. This gives us the ability to encode dependence, e.g. in case two actors are mapped onto the same processor, the ratio of their execution times will always be constant within an iteration.

Fig. 3a shows an example of a SPDF graph where actors have parametric (p, q, s) or constant rates and parametric execution times (a, b, c, d, e). Parametric rates p and s are modified by the actor A every time it fires, while the parametric rate q is modified by the actor B every p^{th} time it fires.

Now we can define our parametric SADF model, by subjecting SPDF to the operational semantics of SADF.

Definition 3. *A parametric rate and parametric actor execution time SADFG (PSADFG) is a tuple* $PSADFG = (G, \Omega, F)$*, where:*

- G *is a live SPDFG;*

- $\Omega = \{\vec{p} \mid \vec{p} \in \mathbb{N}^{+|\mathscr{P}\mathscr{R}|} \times \mathbb{R}_0^{+|\mathscr{P}\mathscr{D}|}\}$ *is a bounded and closed set of all allowed parameter values (rates and actor execution times) for G or shortly the parameter space;*
- $F = (Q, q_0, \delta, \Sigma)$ *is the scenario state transition system consisting of a possibly infinite set Q of states, an initial state* $q_0 \in Q$*, a transition relation* $\delta \subseteq Q \times Q$ *and a scenario labelling* $\Sigma : Q \to \Omega$*.*

In contrast to SADF, which explicitly defines scenarios as a finite collection of SDF graphs, in PSADF scenarios are implicitly defined over the bounded and closed vector parameter space Ω. Elements of Ω are vectors $\vec{p} \in \mathbb{N}^{+|\mathscr{P}\mathscr{R}|} \times \mathbb{R}_0^{+|\mathscr{P}\mathscr{D}|}$. Let $\mathbf{G}(\vec{p})$ be the PSADF (max,+) characteristic $n \times n$ matrix for the parameter space point $\vec{p}$, where n is the number of initial tokens in PSADFG. The operational semantics of the model is as follows: every finite path of arbitrary length $\overline{q}$ over the scenario transition system F corresponds to a sequence $\overline{s}$ with $\overline{s}(k) = \Sigma(\overline{q}(k))$. This is a sequence of parameters space points, i.e. $\overline{s} = \overline{\vec{p}}$. The evaluation of the PSADFG's SPDFG G at a parameter space point is nothing else but an SDFG. The characteristic (max,+) matrix of this SDFG equals to $\mathbf{G}(\vec{p})$ (evaluation at a concrete $\vec{p} \in \Omega$). When the scenario state transition system performs a transition, the SDFG obtained by the evaluation of the PSADFG at that exact point is executed for exactly one iteration. Given previous reasoning, the analogy to SADF is obvious. We can say that PSADF is a compact representation of SADF. From the performance analysis perspective, by using the provision of an infinite (max,+) automaton [11] we can define the completion time of a k-long sequence of parameter point activations as a sequence of (max,+) matrix multiplications $\mathscr{A}(\overline{\vec{p}}) = \mathbf{G}(\vec{p}_k) \ldots \mathbf{G}(\vec{p}_1)\vec{i}$ as it is done in [12] for SADF. The worst case increase of $\mathscr{A}(\overline{\vec{p}})$ for a growing length of $\overline{\vec{p}}$ represents the worst-case throughput for any sequence of parameters points allowed by the scenario transition system.

As already mentioned, PSADF is a compact representation of SADF. We use it to model the behaviour of applications characterized by vast number of scenarios where it is impossible to determine the scenario occurrence pattern even if such exists. Therefore, in terms of PSADF we will be considering the case of a fully connected scenario state transition system, i.e. $\delta = Q \times Q$, and where every state of the transition system corresponds to one parameter space point, i.e. there is a bijective mapping $z : Q \to \Omega$. This way we will always be able to give a conservative bound on the worst-case throughput. This is due to the simple fact that the language recognized by an arbitrary PSADF F is always included in the language recognized by the PSADF F where $\delta = Q \times Q$ and there exists a bijection $z : Q \to \Omega$.

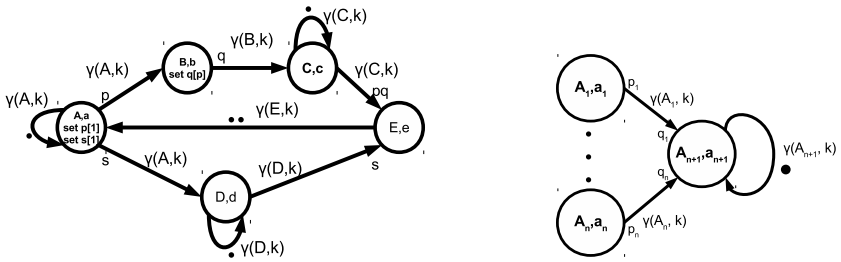
Proposition 1. *The worst-case throughput of a PSADFG for which* $\delta = Q \times Q$ *and for which exists a bijective mapping* $z : Q \to \Omega$ *equals to the inverse of the maximum cycle mean of the MPAG defined by the matrix* $\mathbf{G} = \max_{q \in Q}(\mathbf{G}(z(q)))$.

Proof. Given the operational semantics of PSADF previously described and the fact that Ω is bounded and closed, it follows straightforwardly from [12][11]. □

4.2 Worst-Case Throughput Analysis

4.2.1 Problem Definition.

Given $\mathbf{G}(\vec{p}) = \{g_{ij}(\vec{p})\}$ as a matrix of continuous function over the closed and bounded parameter space Ω that possesses an appropriate mathematical formulation, e.g. as equalities and inequalities over a certain $(|\mathscr{P}\mathscr{R}| + |\mathscr{P}\mathscr{D}|)$-dimensional vector space, using Proposition 1, our worst-case throughput calculation problem becomes a set of maximally $(n \times n)$ constrained optimization problems with $\mathbf{G}(\vec{p}) = \{g_{ij}(\vec{p})\}$ as the objective function(s) and Ω as the constraint set:



(a) An example SPDFG/PSADFG. (b) PSADF actor model.

Figure 3: Parametric SADF

foreach (i, j) s.t. $g_{ij}(\vec{p}) \neq -\infty$ **do**

$$\begin{aligned} \underset{\vec{p}}{\text{maximize}} \quad & g_{ij}(\vec{p}) \\ \text{subject to} \quad & \vec{p} \in \Omega. \end{aligned}$$

A continuous function over a bounded and closed set admits a maximum. Of course, the term continuous includes also discrete functions that are continuous in the Heine sense. After maximizing all the element functions of $\mathbf{G}(\vec{p})$, the worst-case throughput will equal to the MCM of the MPAG given by the maximized PSADFG (max,+) characteristic matrix. Our main challenge is thus to derive a technique for the analytical formulation of the symbolic PSADFG (max,+) characteristic matrix $\mathbf{G}(\vec{p})$. $\mathbf{G}(\vec{p})$ is a matrix of functions that in the (max,+) sense encodes the time distances between initial tokens in adjacent iterations of a PSADFG. We will show that this is a matrix of polynomial functions of $\vec{p}$. Polynomial functions are continuous. Then the problem can be solved as a polynomial programming problem over Ω. There exists a variety of techniques for solving such problems depending on the 'shape' of Ω. Do note here that these optimization problems are solved independently as we are interested in the worst-case increase of $\mathscr{A}(\overline{\vec{p}})$ for a growing length of $\overline{\vec{p}}$ (over a growing number of iterations).

4.2.2 (max,+) Algebra for PSADF.

In PSADF we only allow parameters to change between graph iterations, i.e $\frac{\#M(pr_j)}{\alpha(pr_j)} = 1$ for parametric rates in the context of SPDF. The same goes for parametric actor execution times. Currently, our $\mathbf{G}(\vec{p})$ extraction technique requires that the considered PSADFG is 'acyclic within an iteration'. If we take a PSADFG and convert it to a directed acyclic graph (PSADFG-DAG) by removing the edges with initial tokens, we require that only the PSADFG-DAG sink actors can produce tokens on the removed edges, and only the PSADFG-DAG source actors can consume from those edges. We do not include self-edges in this restriction. That is to say that we only allow cyclic dependencies tied to one actor. However, we can still consider PSADFGs that are serial compositions of subgraphs that are 'acyclic within an iteration' if the subgraph performs only one iteration during an iteration of the composite PSADFG. Our $\mathbf{G}(\vec{p})$ extraction process will depend on the PSADFG quasi-static schedule which can be obtained using the procedure from [10]. Basically, the PSADFG-DAG is sorted topologically. Result of the topological sorting is a string of actors. For PSADFG in Fig. 3a this string equals to $ABCDE$. Now we replace every actor X with $X^{\#X}$, where $\#X$ is the PSADFG repetition vector entry for actor X. For PSADFG in Fig. 3a the final quasi-static schedule takes the form $AB^pC^{pq}D^sE$.

We continue by giving an appropriate (max,+) model of the PSADF actor as displayed in Fig. 3b.

First let us briefly explain the (max,+) semantics of a dataflow actor firing. If T is the set of tokens needed by an actor to perform its firing and for every $\tau \in T$, t_τ is the time that token becomes available, then the starting time of the actor firing is given by $\bigoplus_{\tau \in T} t_\tau$. If d is the execution time of that actor then the tokens produced by the actor firing become available at $\bigoplus_{\tau \in T} t_\tau + d$. Now, let $\gamma(A_i,k)$ be the completion time of the k^{th} firing of actor A_i. This annotation is present in Fig. 3a for each of the actors. In order for an actor to fire, it must have all its input dependencies satisfied. We can now derive the expression for γ:

$$\gamma(A_i,k) = \left(\bigoplus_{A_h \mid (A_h,A_i) \in \mathscr{E}} \gamma\left(A_h, \left\lceil \frac{r(A_i,(A_h,A_i))k - i(A_h,A_i)}{r(A_h,(A_h,A_i))} \right\rceil\right)\right) \otimes e(A_i). \tag{1}$$

The completion time of the k^{th} firing of actor A_i corresponds to the maximal completion times of appropriately indexed firings of actors that feed its input edges $A_h \mid (A_h,A_i) \in \mathscr{E}$ increased by its own execution time $e(A_i)$. The quotient $\left\lceil \frac{r(A_i,(A_h,A_i))k-i(A_h,A_i)}{r(A_h,(A_h,A_i))} \right\rceil$ is used to index the appropriate firing of the actors that feed its input edges. The $i(A_h,A_i)$ member in the nominator of the fraction accounts for initial tokens. Initial tokens have the semantics of the initial delay and form the initial conditions used to solve (max,+) difference equations, analogue to the initial conditions in classical linear difference (recurrence) equations. We comply with the liveness criteria from [10] which among others requires that all SPDFG cycles are live, i.e. within a cycle there is an edge with initial tokens to fire the actor the needed number of times to complete an iteration, either a global one or a local one. Liveness and the 'acyclic within an iteration' restriction render (1) solvable and we can always obtain a solution for (1) in terms of initial conditions. The analytical solution of a system of such (max,+) linear difference equations evaluated at the iteration boundary for every actor of the graph will exactly give us the needed symbolic PSADFG characteristic (max,+) matrix. We follow the order of actors from the quasi-static schedule. This guarantees that we respect data/resource dependencies. Element $X^{\#X}$ tells us that we have to solve (1) for actor X at $k = \#X$. The obtained solution is propagated to the next iteration of the algorithm. We continue until we reach the end of the quasi-static schedule. At this point we will obtain solutions for all actors in terms of dependence of their completion times at the iteration boundary on initial conditions. From these solutions we can then easily construct the symbolic PSADFG characteristic (max,+) matrix.

Let us consider the PSADFG example in Fig. 3a. We write down (max,+) equations for each actor (we omit the sign $\otimes$, i.e. $a \otimes b$ will be denoted as ab):

$$\gamma(A,k) = (\gamma(A,k-1) \oplus \gamma(E,k-2))\, a = a\gamma(A,k-1) \oplus a\gamma(E,k-2), \tag{2}$$

$$\gamma(B,k) = b\gamma(A, \lceil \frac{k}{p} \rceil), \tag{3}$$

$$\gamma(C,k) = \left(\gamma(B, \lceil \frac{k}{q} \rceil) \oplus \gamma(C,k-1)\right) c = c\gamma(B, \lceil \frac{k}{q} \rceil) \oplus c\gamma(C,k-1), \tag{4}$$

$$\gamma(D,k) = \left(\gamma(A, \lceil \frac{k}{s} \rceil) \oplus \gamma(D,k-1)\right) d = d\gamma(A, \lceil \frac{k}{s} \rceil) \oplus d\gamma(D,k-1), \tag{5}$$

$$\gamma(E,k) = (\gamma(C,pqk) \oplus \gamma(D,sk))\, e = e\gamma(C,pqk) \oplus e\gamma(D,sk). \tag{6}$$

The initial conditions are:

$$\gamma(A,0) = t_1,\ \gamma(D,0) = t_2,\ \gamma(C,0) = t_3,\ \gamma(E,-1) = t_4,\ \gamma(E,0) = t_5. \tag{7}$$

We can now evaluate and solve them at an iteration boundary given by the sequential schedule $AB^pC^{pq}D^sE$. Firing actor A using (2) with $k = 1$ we obtain:

$$\gamma(A,1) = a\gamma(A,0) \oplus a\gamma(E,-1) = at_1 \oplus at_4. \tag{8}$$

Firing B^p using (3) with $k = p$ and using (8) we obtain:

$$\gamma(B,p) = abt_1 \oplus abt_4. \tag{9}$$

Firing C^{pq} using (4) with $k = pq$ and (9) we obtain (backward substitution):

$$\gamma(C,pq) = abct_1 \oplus abct_4 \oplus c\gamma(C,pq-1) = abc^{pq}t_1 \oplus c^{pq}t_3 \oplus abc^{pq}t_4. \tag{10}$$

Firing D^s using (5) with $k = s$ similarly evaluates to:

$$\gamma(D,s) = ad^st_1 \oplus d^st_2 \oplus ad^st_4. \tag{11}$$

Firing E using (6) with $k = 1$ and (10) (11) we obtain:

$$\gamma(E,1) = aet_1(bc^{pq} \oplus d^s) \oplus d^set_2 \oplus c^{pq}et_3 \oplus aet_4(bc^{pq} \oplus d^s). \tag{12}$$

In (12) initial conditions t_1 and t_4 are (max,+) multiplied by a symbolic (max,+) summation term $(bc^{pq} \oplus d^s)$. We refer to this situation as a *conflict*. The production time of the tokens generated by actor E will depend on the relationship between $(b+pqc)$ and sd. Before proceeding, we have to consider two cases. One given by $(b+pqc \geq sd)$ and the other by $(b+pqc \leq sd)$. We must check the intersection of newly added constraints and the already existing ones to reason against feasibility. If there are no feasible points in one of the subregions, we drop the further evaluation within the same subregion. In this example let us assume that both subregions contain feasible points. We easily construct the symbolic matrices from the solutions that are all expressed in terms of their dependence on initial conditions at an iteration boundary. We write down once more the solutions of the equations at the iteration boundary for actors that reproduce the initial tokens. Those are actors (A,C,D,E). We will change the notation from $\gamma(A_i,k)$ to t'_j depending on the indexes of initial conditions (tokens) and the producing actor. We obtain for $(b+pqc \geq sd)$:

$$t'_1 = at_1 \oplus at_4, \tag{13}$$
$$t'_2 = ad^st_1 \oplus d^st_2 \oplus ad^st_4, \tag{14}$$
$$t'_3 = abc^{pq}t_1 \oplus c^{pq}t_3 \oplus abc^{pq}t_4, \tag{15}$$
$$t'_4 = t_5, \tag{16}$$
$$t'_5 = abc^{pq}et_1 \oplus d^set_2 \oplus c^{pq}et_3 \oplus abc^{pq}et_4. \tag{17}$$

From (13)-(17) we then easily obtain the rows of the symbolic *(max,+)* matrix:

$$\mathbf{G}_{(b+pqc \geq sd)} = \begin{bmatrix} a & -\infty & -\infty & a & -\infty \\ a+sd & sd & -\infty & a+sd & -\infty \\ a+b+pqc & -\infty & pqc & a+b+pqc & -\infty \\ -\infty & -\infty & -\infty & -\infty & 0 \\ a+b+pqc+e & sd+e & pqc+e & a+b+pqc+e & -\infty \end{bmatrix}.$$

The same procedure is used for the $(b+pqc \leq sd)$ case. The evolution of the PSADF graph is then governed by the following equations over the parameter space Ω: $\vec{\gamma}_{k+1} = \mathbf{G}_{(b+pqc \geq sd)}\vec{\gamma}_k$ and $\vec{\gamma}_{k+1} = \mathbf{G}_{(b+pqc \leq sd)}\vec{\gamma}_k$, depending in which region of Ω is the $(k+1)^{\text{th}}$ iteration scheduled. If $(b+pqc = sd)$, any of the two can be chosen. In the definition of both regions we use the $\leq$ and $\geq$ operators to have them remain closed. The functions that constitute the symbolic (max,+) matrices are polynomial functions of $\vec{p}$.

In order to obtain the worst case throughput we will have to solve a mixed-integer polynomial programming problem for $\mathbf{G}_{(b+pqc \geq sd)}$ and $\mathbf{G}_{(b+pqc < sd)}$ over $(\Omega \cap (b+pqc \geq sd))$ and $(\Omega \cap (b+pqc \leq sd))$, respectively. A collection of techniques that solve such problems for a variety of definitions of Ω, e.g. convex, non-convex or restricted to take only a few discrete values, can be found in [21]. The matrix $max\left(\mathbf{G}_{(b+pqc \geq sd)}, \mathbf{G}_{(b+pqc \leq sd)}\right)$ will define the MPAG of the example PSADFG. The inverse of the MCM of this MPAG equals to the worst-case throughput.

At this point we present our recursive algorithm for symbolic PSADF (max,+) characteristic matrix extraction (Algorithm 1). The inputs to the algorithm are the pre-computed sequential quasi-static sched-

Algorithm 1 Symbolic PSADFG (max,+) characteristic matrix extraction

```
1:  function SymbolicExtract(Qss, MpEqSet, Φ, Ss)
2:      fBranchingNode ← false
3:      while not Qss.isFinished() do
4:          currQssElem ← Qss.popNextElem()
5:          currSol ← Solve(MpEqSet, currQssElem)
6:          if currSol.Conflicted() then
7:              fBranchingNode ← true
8:              while newΦ ← currSol.getNextConflict() do
9:                  if FeasibilityCheck(newΦ, Φ) then
10:                     currΦ ← Φ
11:                     currΦ.Add(newΦ)
12:                     currMpEqSet ← MpEqSet
13:                     currMpEqSet.ResolveC(newΦ)
14:                     Ss.Add(SymbolicExtract(Qss, currMpEqSet, currΦ, Ss))
15:                 end if
16:             end while
17:         else
18:             MpEqSet.Update(currSol)
19:         end if
20:     end while
21:     if not fBranchingNode then
22:         return (mpEqSet, Φ)
23:     else
24:         return ∅
25:     end if
26: end function
```

ule *Qss*, the set of PSADF (max,+) difference equations *MpEqSet*, the initial parameter space $\Phi = \Omega$ and the initial solution set $Ss = \emptyset$. The solution set *Ss* is a set of ordered pairs $Ss = \{(\mathbf{G}_{\Phi_i}, \Phi_i)\}$, where $\mathbf{G}_{\Phi_i}$ is the symbolic (max,+) matrix that governs the evolution of the PSADF in the region $\Phi_i \subseteq \Omega$ generated by

adding conflict resolving constraints to Ω during the execution of the algorithm. Algorithm traverses the sequential schedule taking one actor with its repetition count at a time (Line 3). Function SOLVE (Line 5) solves Equation (1) for the considered actor. If there are no conflicts in the solution, the algorithm updates the equation set with the current solution that can be used in later iterations (Line 18). If there are conflicts, i.e. there are $\bigoplus_i y_i$ terms multiplying the initial conditions, we have to split the parameter space (Line 8). For example, if the term $y_1 \oplus y_2 \oplus y_3$ is multiplying an initial condition, we have to consider three cases: $(y_1 > y_2, y_1 > y_3)$, $(y_2 > y_1, y_2 > y_3)$ and $(y_3 > y_1, y_3 > y_2)$. Function FEASIBILITYCHECK (Line 9) checks the emptiness of the intersection of the current constraint set Φ and the new constraints. If the intersection is non-empty, new constraints are added to the current set for this branch of exploration (Line 11), conflicts are resolved (Line 13) and SYMBOLICEXTRACT is recursively called again (Line 14). If the intersection is non-feasible, this branch is dropped. If we continue in this fashion we will eventually reach a non-branching node (Line 22).

We demonstrate our approach on the example PSADF graph in Fig. 3a. The example models a dynamic streaming application consisting of loops with interdependent parametric affine loop bounds. We define the ranges for parametric loop bounds (PSADF rates) as: $p \in [10, 2000], q \in [10, 15]$ and $s \in [100, 1500]$. We also define linear dependencies between them: $p + s \leq 1400$ and $q \leq p$. Our application is run on a multi-processor platform where each loop body (actor) is mapped onto a different processor. Let PSADF actor execution times take the values of their nominal execution times multiplied by the parameter $c_i \in [1, 5]$ to account for six different possible platform dynamic voltage and frequency scaling (DVFS) settings. We obtain: $a = 30c_i$, $b = 20c_i$, $c = 4c_i$, $d = 3c_i$, $e = c_i$. These constraints define Ω for our example. To obtain the worst-case throughput value we must maximize the matrices $\mathbf{G}_{(b+pqc \geq sd)}$ and $\mathbf{G}_{(b+pqc \leq sd)}$ over Ω as given by the previously listed constraints. These become two mixed integer polynomial programming problems over $\Omega \cap (b + pqc \geq sd)$ and $\Omega \cap (b + pqc \leq sd)$ and can be solved using the technique from [21]. Throughput is given by the inverse of the MCM of the MPAG defined by the matrix $max\left(\mathbf{G}_{(b+pqc \leq sd)}, \mathbf{G}_{(b+pqc \geq sd)}\right)$ and equals to $1/390000$ iterations per time-unit.

5 Experimental results

We demonstrate our throughput analysis technique on five representative DSP applications with parametric interdependent affine loop bounds listed in Table 1. The first column shows the number of PSADFG actors, the second denotes the number of initial tokens, the third shows the number of parametric rates, the fourth gives the number of parametric actor execution times and the last shows the number of scenarios as the number of points in the PSADFG parameter space Ω. All applications, except the bounded block parallel lattice reduction algorithm for MIMO-OFDM [3], are mapped onto a two-processor scalar architecture. The latter is mapped onto a vector/SIMD architecture. To obtain the nominal actor execution times for our benchmark set, we used the AVR32 [1] simulator under a reference frequency of 32 MHz. For bounded block parallel lattice reduction algorithm [3] we used random numbers for nominal actor execution times, as the source code of the algorithm is not publicly available. We assume that the frequency of each platform processor can be placed inside the range from 32 to 64 MHz, with the step of 1 Mhz. For a 2 processor platform this will give 32 possible combinations. In contrast to the conventional SADF approach from [12] which would have to generate $|\Omega|$ SDFGs, our approach in each of these cases will solve maximally $(n \times n)$ polynomial programming problems without the need for the enumeration of Ω which is a difficulty by itself. Actually, in practice this number is usually less than $(n \times n)$, because not all initial tokens depend on all other initial tokens in the graph rendering the matrices to be quite sparse. Moreover, sometimes the entries in the symbolic PSADF (max,+) characteristic

Table 1: Experimental results

Benchmark	$\lvert\mathscr{A}\rvert$	n	$\lvert\mathscr{PR}\rvert$	$\lvert\mathscr{PD}\rvert$	$\lvert\Omega\rvert$
Fundam. freq. detector based on norm. autocorr. [2]	12	6	2	2	$16,687,681 \cdot 32$
Normalized LMS alg. [2]	9	6	2	2	$385 \cdot 32$
High resolution spectral analysis [2]	9	6	2	2	$385 \cdot 32$
Adaptive predictor program [7]	6	4	2	2	$400 \cdot 32$
Bound. block parallel latt. reduct. alg. [3]	12	5	3	1	$300 \cdot 16$

matrix are repetitive, so we only have to solve the corresponding problem once. The symbolic PSADF (max,+) characteristic matrices of the benchmark applications were extracted manually using Algorithm 1, while the corresponding optimization problems were solved using CVX, a package for specifying and solving convex programs [17][16].

6 Conclusion

In this paper we have presented an extension to SADF that allows to model applications with vast or infinite sets of behaviours. We refer to our model as PSADF. We have proven the semantical equivalence of the two models and used that result in the formulation of worst-case throughput calculation problem for PSADF graphs with a fully connected state transition system within a generic optimization framework. The objective functions are functionals that represent the elements of the symbolic PSADF (max,+) characteristic matrices. Furthermore, we have derived a (max,+) linear theory based algorithm that is able to generate these matrices by combining a (max,+) difference equation solver and a recursive parameter space exploration for a subclass of PSADF graphs that are 'acyclic within an iteration'. As future work, we want to fully automate our technique and investigate the problem of parametric throughput analysis of PSADF graphs.

References

[1] *Atmel AVR*. Available at `http://www.atmel.com/images/doc32000.pdf`.

[2] *ICST Signal Processing Library Ver. 1.2*. Available at `http://www.icst.net/research/projects/dsp-library/`.

[3] U. Ahmad, Min Li, S. Pollin, R. Fasthuber, L. Van der Perre & F. Catthoor (2010): *Bounded Block Parallel Lattice Reduction algorithm for MIMO-OFDM and its application in LTE MIMO receiver*. In: *Signal Processing Systems (SIPS), 2010 IEEE Workshop on*, pp. 168–173, doi:10.1109/SIPS.2010.5624784.

[4] Franois Baccelli, Guy Cohen, Geert Jan Olsder & Jean-Pierre Quadrat (1992): *Synchronization and linearity: an algebra for discrete event systems*. John Wiley & Sons, Inc.

[5] B. Bhattacharya & S.S. Bhattacharyya (2000): *Parameterized dataflow modeling of DSP systems*. In: *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, 6, pp. 3362–3365 vol.6, doi:10.1109/ICASSP.2000.860121.

[6] G. Bilsen, M. Engels, R. Lauwereins & J. Peperstraete (1996): *Cycle-static dataflow*. *Signal Processing, IEEE Transactions on* 44(2), pp. 397–408, doi:10.1109/78.485935.

[7] Rulph Chassaing (1999): *Digital Signal Processing: Laboratory Experiments Using C and the TMS320C31 DSK*, 1st edition. John Wiley & Sons, Inc., New York, NY, USA.

[8] Philippe Clauss & Vincent Loechner (1998): *Parametric Analysis of Polyhedral Iteration Spaces*. *Journal of VLSI signal processing systems for signal, image and video technology* 19(2), pp. 179–194, doi:10.1023/A:1008069920230.

[9] M. Damavandpeyma, S. Stuijk, M. Geilen, T. Basten & H. Corporaal (2012): *Parametric throughput analysis of scenario-aware dataflow graphs*. In: *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pp. 219–226, doi:10.1109/ICCD.2012.6378644.

[10] P. Fradet, A. Girault & P. Poplavko (2012): *SPDF: A schedulable parametric data-flow MoC*. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pp. 769–774, doi:10.1109/DATE.2012.6176572.

[11] S. Gaubert (1995): *Performance evaluation of (max,+) automata*. *Automatic Control, IEEE Transactions on* 40(12), pp. 2014–2025, doi:10.1109/9.478227.

[12] M. Geilen & S. Stuijk (2010): *Worst-case performance analysis of Synchronous Dataflow scenarios*. In: *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pp. 125–134.

[13] Marc Geilen (2011): *Synchronous Dataflow Scenarios*. *ACM Trans. Embed. Comput. Syst.* 10(2), pp. 16:1–16:31, doi:10.1145/1880050.1880052.

[14] A.-H. Ghamarian, M. C W Geilen, T. Basten & S. Stuijk (2008): *Parametric Throughput Analysis of Synchronous Data Flow Graphs*. In: *Design, Automation and Test in Europe, 2008. DATE '08*, pp. 116–121, doi:10.1109/DATE.2008.4484672.

[15] A.-H. Ghamarian, M. C W Geilen, S. Stuijk, T. Basten, A. J M Moonen, M.J.G. Bekooij, B.D. Theelen & M.R. Mousavi (2006): *Throughput Analysis of Synchronous Data Flow Graphs*. In: *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, pp. 25–36, doi:10.1109/ACSD.2006.33.

[16] Michael Grant & Stephen Boyd (2008): *Graph Implementations for Nonsmooth Convex Programs*. In: *Recent Advances in Learning and Control, Lecture Notes in Control and Information Sciences* 371, Springer London, pp. 95–110, doi:10.1007/978-1-84800-155-8_7.

[17] Michael Grant & Stephen Boyd (2013): *CVX: Matlab Software for Disciplined Convex Programming, version 2.0 beta*. Available at http://cvxr.com/cvx.

[18] Gilles Kahn (1974): *The Semantics of Simple Language for Parallel Programming*. In: *IFIP Congress*, pp. 471–475.

[19] E.A. Lee & D.G. Messerschmitt (1987): *Synchronous data flow*. *Proceedings of the IEEE* 75(9), pp. 1235–1245, doi:10.1109/PROC.1987.13876.

[20] P. Macken, M. Degrauwe, M. Van Paemel & H. Oguey (1990): *A voltage reduction technique for digital systems*. In: *Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC., 1990 IEEE International*, pp. 238–239, doi:10.1109/ISSCC.1990.110213.

[21] Hanif D. Sherali & W.P. Adams (1998): *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Available at http://www.springer.com/mathematics/book/978-0-7923-5487-1.

[22] B.D. Theelen, M. C W Geilen, T. Basten, J. P M Voeten, S. V. Gheorghita & S. Stuijk (2006): *A scenario-aware data flow model for combined long-run average and worst-case performance analysis*. In: *Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings. Fourth ACM and IEEE International Conference on*, pp. 185–194, doi:10.1109/MEMCOD.2006.1695924.

[23] Maarten Hendrik Wiggers (2009): *Aperiodic Multiprocessor Scheduling for Real-Time Stream Processing Applications*. Ph.d. dissertation, doi:10.3990/1.9789036528504.