Uli Fahrenberg
Stavros Tripakis (Eds.)

LNCS 6919

Formal Modeling and Analysis of Timed Systems

9th International Conference, FORMATS 2011
Aalborg, Denmark, September 2011
Proceedings

Springer

Lecture Notes in Computer Science 6919

Uli Fahrenberg   Stavros Tripakis (Eds.)

Formal Modeling and Analysis of Timed Systems

9th International Conference, FORMATS 2011
Aalborg, Denmark, September 21-23, 2011
Proceedings

Springer

Volume Editors

Uli Fahrenberg
IRISA/INRIA Rennes
Campus Beaulieu, 35042 Rennes Cedex, France
E-mail: ulrich.fahrenberg@irisa.fr

Stavros Tripakis
University of California
Berkeley, CA 94720-1770, USA
E-mail: stavros.tripakis@gmail.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-24309-7 e-ISBN 978-3-642-24310-3
DOI 10.1007/978-3-642-24310-3
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011936636

CR Subject Classification (1998): F.3, D.2, D.3, D.2.4, F.4.1, C.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

It is our pleasure and honor to present the proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2011), held during September 21–23, 2011 in Aalborg, Denmark.

Timing aspects of systems have been traditionally treated by researchers interested in a variety of topics, such as semantics, verification, performance analysis, digital circuit design, and embedded systems. Timing-related questions studied in these communities do have their particularities. However, there is a growing awareness that there are basic problems that are common to all of them. In particular, all these sub-disciplines treat systems whose behavior depends on combinations of logical and temporal constraints; namely, constraints on the temporal distances between occurrences of events. The aim of FORMATS is to promote the study of fundamental and practical aspects of timed systems, and to bring together researchers from different disciplines that share interests in modeling and analysis of timed systems.

This year FORMATS received 43 submissions, each of which was reviewed by at least three Program Committee members. Based on the reviews and discussions that ensued, we selected 20 papers for publication and presentation at the conference. The program also included three invited talks:

– Rajeev Alur, University of Pennsylvania, Philadelphia: *Interfaces for Control Components*
– Boudewijn Haverkort, University of Twente, Enschede, and Embedded Systems Institute, Eindhoven: *Formal Modeling and Analysis of Timed Systems: Technology Push or Market Pull?*
– Oded Maler, CNRS, VERIMAG Laboratory, Grenoble: *Performance Evaluation of Schedulers in a Probabilistic Setting*

We would like to thank all authors for submitting to FORMATS. We also wish to thank the invited speakers for accepting our invitation and providing abstracts or full papers for the conference proceedings. We are especially grateful to the members of the Program Committee and other reviewers for their hard work which made this conference possible. Their competent and timely handling of the submissions is greatly appreciated.

We would also like to thank the local organizers, Alexandre David, Kim Larsen, Claus Thrane, and Rikke Uhrenholt, for arranging an excellent conference. We thank the providers of the EasyChair conference management system, which has been of great value, and the Springer LNCS team for their excellent

support. Finally, we gratefully acknowledge the financial support we received from the Danish VKR Centre of Excellence MT-LAB, from the European Network of Excellence ArtistDesign, and from the Centre for Embedded Software Systems CISS.

July 2011

Uli Fahrenberg
Stavros Tripakis

Organization

Program Chairs

Uli Fahrenberg	IRISA/INRIA Rennes, France
Stavros Tripakis	University of California, Berkeley, USA

Program Committee

Christel Baier	Technical University of Dresden, Germany
Patricia Bouyer	LSV, CNRS and ENS Cachan, France
Marius Bozga	Verimag/CNRS, Grenoble, France
Víctor Braberman	Universidad de Buenos Aires, Argentina
Luboš Brim	Masaryk University, Brno, Czech Republic
Krishnendu Chatterjee	Institute of Science and Technology (IST), Austria
Deepak D'Souza	Indian Institute of Science, Bangalore, India
Manfred Droste	Leipzig University, Germany
Uli Fahrenberg	IRISA/INRIA Rennes, France
Martin Fränzle	Carl von Ossietzky Universität Oldenburg, Germany
Marc Geilen	Eindhoven University of Technology, The Netherlands
Radu Grosu	Stony Brook, USA
Holger Hermanns	Saarland University, Germany
Bengt Jonsson	Uppsala University, Sweden
Kim G. Larsen	Aalborg University, Denmark
Axel Legay	IRISA/INRIA Rennes, France
Jean-François Raskin	Université Libre de Bruxelles, Belgium
Jan Reineke	UC Berkeley, USA
Jiří Srba	Aalborg University, Denmark
Paulo Tabuada	UCLA, USA
Ashish Tiwari	SRI International, USA
Stavros Tripakis	University of California, Berkeley, USA
Ben Worrell	Oxford University, UK

Steering Committee

Rajeev Alur	University of Pennsylvania, USA
Eugene Asarin	Université Paris 7, France
Thomas A. Henzinger	Institute of Science and Technology (IST), Austria

Joost-Pieter Katoen	RWTH Aachen, Germany
Kim G. Larsen	Aalborg University, Denmark
Oded Maler	Verimag, Grenoble, France
Lothar Thiele	ETH Zürich, Switzerland
Wang Yi	Uppsala University, Sweden

Additional Reviewers

Sebastian Altmeyer
Paolo Ballarini
Jiří Barnat
Ezio Bartocci
Sebastian Bauer
Benedikt Bollig
Thomas Brihaye
Véronique Bruyère
Peter Bulychev
Jakub Chaloupka
Alexandre David
Alexandre Donzé
Christian Eisentraut
Georgios Fainekos
Gilles Geeraerts
Madhu Gopinathan
Daniel Grund
Ernst Moritz Hahn
Alexander Heußner
Md Ariful Islam
Kenneth Yrke Jørgensen
Panagiotis Katsaros
Jan Krčál
Marta Kwiatkowska
Kai Lampka
Marius Mikučionis
Raj Mohan M.
Abhishek Murthy
Mikael H. Møller
Dejan Ničković
Andreas Podelski
Pavithra Prabhakar
Vinayak Prabhu
Karin Quaas
Jan-David Quesel
Elaine Render
Pierre-Alain Reynier
Philipp Ruemmer
David Safranek
Ocan Sankur
Sven Schewe
Christian Stehno
Mani Swaminathan
Claus Thrane
Louis-Marie Traonouez
Wang Yi
Andrzej Zbrzezny
Milan Češka

Table of Contents

Invited Talks

Session 1: Probabilistic Models

Session 2: Robustness

Session 3: Games

Session 4: Verification and Testing

Session 5: Verification

Session 6: Hybrid Systems

Session 7: Applications

Performance Evaluation of Schedulers in a Probabilistic Setting

Jean-Francois Kempf, Marius Bozga, and Oded Maler

CNRS-VERIMAG
University of Grenoble
France
Jean-francois.Kempf@imag.fr

Abstract. We show how to evaluate the performance of solutions to finite-horizon scheduling problems where task durations are specified by bounded *uniform* distributions. Our computational technique, based on computing the volumes of zones, constitutes a contribution to the computational study of scheduling under uncertainty and stochastic systems in general.

1 Introduction

Scheduling, the allocation of limited reusable resources over time to competing tasks, is a universal activity. It is performed routinely in domains of very different scales in terms of time, space and energy. These include the allocation of airways and runways to flights, allocating machines to different product lines in a factory, and the efficient allocation of computation and communication resources to information-processing tasks. This latter activity is becoming of prime importance in many scales, ranging from world-wide cloud computing, via the realization of multiple distributed control loops, down to mapping and scheduling tasks on multi-core computers. In all such situations one wants to synthesize schedulers which are optimal or good in some sense, or at least to be able to compare the performance of proposed schedulers and choose the better ones. Performance and optimality of such schedulers are typically based on the quantity of work performed over time, which in the case of a finite amount of work can be expressed as *termination* time. Good schedules are typically associated with intensive, almost idle-free, utilization of critical bottleneck resources.

In a *deterministic* setting one assumes that everything is known *in advance* about the demand for work, including the tasks to be executed, their arrival times and the durations for which they occupy resources. In other words, once the scheduling policy itself is determined, the system admits a *unique* execution scenario (run, realization). Evaluating a scheduler based on this unique run is straightforward – just simulate it – while finding an optimal scheduler for any non-trivial scheduling problem (such as job-shop) is NP-hard or worse. However, determinism is rarely the case in real life and exact duration of tasks, their arrival times and many other features may vary to large extents. Each instance in this uncontrollable space yields a *different schedule* and the overall evaluation of a scheduler or a scheduling policy, which can be viewed as a strategy in a two-person timed game [7] with uncertainty viewed as an adversary, should be based on some *quantification* over all possible behaviors it induces [28].

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 1–17, 2011.

This adversarial time-optimality problem has been tackled in [6,1] using a *worst-case* approach on models of different types of uncertainty. In [6], using, the general model of *timed game automaton* [7] where the adversary is discrete, the following problem was proved to be decidable: synthesize a controller which is worst-case time-optimal in the sense that the maximal (over all possible runs induced by the adversary) time to reach a goal state is minimal. In [1] the case of job-shop scheduling with uncertain task *durations* each ranging over a bounded interval was treated. For this problem, worst-case optimality is defined trivially by the optimal solution to a *deterministic* scheduling problem associated with the worst case where all tasks take their respective maximal duration. One has to define a new notion of optimality (d-future optimal strategies) to make the optimal synthesis problem meaningful, resulting in a synthesis algorithm based on value iteration over sets of clock valuations (zones) which can be seen as an *offline* version of some kind of *model-predictive control*.

The use of worst-case reasoning is to some extent a residue of the safety-critical banner under which formal verification has been argued for, but in many (if not most) real-life situations, temporal uncertainty is modeled probabilistically as a distribution over the durations of each task and scheduler quality is measured accordingly, for example by the *expected* completion time or by its maximum over all but a small fraction of the runs. In this paper we develop and implement a computational framework in order to compute the performance of such schedulers, modeled by automata similar in structure to those used in [1] but whose durations are probabilistic. Such automata are sufficiently rich to express stochastic variants of well-known scheduling problems such as job-shop or task-graph. Formal definitions of these *duration probabilistic automata* and their semantics can be found in [29].

The study of continuous-time stochastic processes has been going on for many years in other branches of mathematics where simple computational questions like those we pose are not typically asked, as well as in closer domains such as probabilistic verification and performance evaluation [13,11]. A well-studied class of such processes are *continuous-time Markov chains* (CTMC) where durations are distributed *exponentially*. Such distributions are memoryless in the sense that time spent waiting for a task to terminate does not influence the distribution on the remaining time. As a result they are easy to compute with and problems such as model-checking against qualitative [3] and quantitative [8] temporal properties or optimal controller synthesis for finite-horizon problems [1] are well understood. This forgetfulness assumption may be realistic and useful for modeling request arrivals in queuing models, but seems inappropriate for modeling the durations of several instances of the same computational task.[1]

In this paper we assume task durations to be *uniform* over a *bounded* interval, which is a natural "stochastization" of the set-theoretic temporal uncertainty of timed automata. Handling such systems we find ourselves in the realm of the so-called *generalized semi-Markov processes* (GSMP), a class of continuous-time stochastic processes [21,22,15,25]. Similar computational studies of GSMPs include [2,10], [14,30] and [29]. The former are concerned with verifying temporal properties for some classes of GSMPs and develop techniques to determine whether the probability of a

[1] The academic paper industry is perhaps the prime example of an application domain where this hypothesis is useful.

property-violating behavior is zero. The work of [14,30] is concerned with stochastic Petri nets for which a computational framework is developed for propagating densities in the marking graph. This work, as well as [29] on duration probabilistic automata, use densities on clocks which are auxiliary state variables. At each reachable state and zone in the clock space, the distribution over clock values is maintained and used to compute the distribution after the next transition. In contrast, the approach presented in this paper works directly on the space of the *duration* random variables and does not use clocks explicitly. Similar ideas were developed in [24] to compute the probability of test cases in timed systems.

The rest of the paper is organized as follows. Section 2 defines single and parallel processes, their behaviors (timed and qualitative) and presents a useful coordinate transformation between durations and time stamps. Section 3 shows how to derive the timing constraints associated with a qualitative behavior when processes execute independently without resource conflicts, and how to compute the volumes of the polytopes they define. Section 4 extends the framework to the more interesting case of resource conflicts that have to be resolved by dynamic scheduling strategies and presents very preliminary experimental results. A discussion of future directions concludes the paper.

2 Preliminaries

We consider a composition $S = P^{\mathbf{1}}||\cdots||P^{\mathbf{n}}$ of n *sequential stochastic processes*, each consisting of a sequence of steps. Each step has a probabilistic duration and cannot start before its predecessor terminates. We consider two execution frameworks:

1. *Independent execution:* all processes start simultaneously and each process starts a step as soon as its preceding step has terminated, regardless of the state of other processes;
2. *Coordinated execution:* the initiation of a step is controlled by a *scheduler* which may hold a step of one process in a waiting state until the termination of a step of another process that uses the same resource.

The second framework will allow us to compare schedulers but we start with the first because it is simpler, does not require knowledge of timed automata and hence is accessible to a wider audience. On this simpler model we will develop the basic computational machinery that will allow us to compute the probabilities of different *qualitative behaviors*, each corresponding to an equivalence class of timed behaviors associated with a particular *order* in which steps of different processes terminate.

Definition 1 (Uniform Distribution). *A* uniform *distribution inside an interval* $I = [a, b]$ *is characterized by a density* ψ *defined as*

$$\psi(y) = \begin{cases} 1/(b-a) & \textit{if}\, a \leq y < b \\ 0 & \textit{otherwise} \end{cases}$$

and in terms of distribution as

$$F(y) = \int_0^y \psi(\tau)d\tau = \begin{cases} 0 & \textit{if}\, y < a \\ (y-a)/(b-a) & \textit{if}\, a \leq y \leq b \\ 1 & \textit{if}\, b \leq y \end{cases}$$

Definition 2 (Process). *A sequential stochastic process is a pair $P = (\mathcal{I}, \Psi)$ where $\mathcal{I} = \{I_j\}_{j \in K}$ is a sequence of duration intervals and $\Psi = \{\psi_j\}_{j \in K}$ is a matching sequence of densities with ψ_j being the uniform density over $I_j = [a_j, b_j]$, indicating the duration of step j.*

We consider *finite* processes with $K = \{1, \ldots, k\}$. Probabilistically speaking, step durations can be viewed as a finite sequence of *independent* uniform random variables $\{y_j\}_{j \in K}$ that we denote as vectors $y = (y_1, \ldots, y_k)$ ranging over a *duration space*

$$D = I_1 \times \cdots \times I_k \subseteq \mathbb{R}^k$$

with density $\psi(y_1, \ldots, y_k) = \psi_1(y_1) \cdots \psi_k(y_k)$. Each point y in the duration space induces a *unique* behavior of the system written as a *time-event sequence* of the form

$$\xi_y = y_1\, e_1\, y_2\, e_2\, \cdots\, y_k\, e_k. \tag{1}$$

Time event sequences are alternations between time elapses represented by real numbers and discrete events that take no time. In the case of a single process $y_j \in I_j$ is the *duration* of step j and e_j is the *event* of terminating that step. The *timed language*[2] associated with the process consists of all the timed behaviors it may generate, namely $L = \{\xi_y : y \in D\}$. The *untimed language* associated with the process is $\underline{L}$, obtained by projecting away durations and retaining events and their order. In the case of a single process $\underline{L}$ is simply the singleton language $\{w\}$ where $w = e_1\, e_2 \cdots\, e_k$.

Mechanically speaking the process behaviors can be viewed as generated by the automaton of Fig. 1 in which being at state q_j corresponds to executing step j. Each run of the automaton is associated with a point y in the duration space. Upon entering q_j an auxiliary clock variable x is reset to zero and the termination transition labeled by e_j is taken exactly when $x = y_j$.

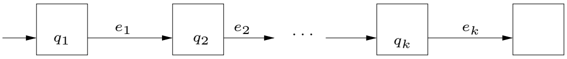


Fig. 1. An automaton view of a process

Suppose we want to characterize the probability of a certain subset of L. For example those behaviors in which for every j the actual duration of step j is in some sub-interval $I'_j = [a'_j, b'_j] \subseteq I_j$. The total probability of these behaviors is simply the *volume* of the rectangle $I'_1 \times \cdots \times I'_k$ divided by the volume of the whole rectangle D. Probabilities of other subsets of the language can be more interesting but harder to compute. For example, the probability that the whole process terminates before some deadline r is simply the volume of the subset of D satisfying $y_1 + \cdots + y_k < r$ divided by the volume of D. Our technique is based on computing such volumes for a system of several parallel processes as described in the sequel.

[2] In the computer science tradition the term *language* is often used to denote a set of sequences or other objects that define dynamic behaviors.

It turns out to be easier to compute volumes after a coordinate transformation from the space of durations to the space of *time stamps* consisting of vectors $t = (t_1, \ldots, t_k)$ where t_j is the absolute occurrence time of event e_j, defined as $t_j = y_1 + y_2 + \cdots + y_j$. A behavior ξ_y can thus be written also as a sequence of time-stamped events[3]

$$\xi_t = (e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k).$$

Assuming that all durations admit a positive lower bound $a_j > 0$, all time stamps satisfy *precedence constraints* of the form $t_j < t_{j+1}$.

Converting y to t and vice versa is done by the linear transformations $t = Ty$ and $y = T't$ where T and T' are matrices of the form

$$T = \begin{pmatrix} 1\,0\,0\,0 \\ 1\,1\,0\,0 \\ 1\,1\,1\,0 \\ 1\,1\,1\,1 \end{pmatrix} \quad T' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

These matrices are lower triangular (the value of t_j cannot depend on a duration $y_{j'}$ with $j' > j$) and their diagonal entries are equal to 1. The determinant of a triangular matrix is equal to the product of the diagonal entries which is 1 and hence the transformations are *volume preserving*. This means that the volume of the duration space D is equal to the volume of the *time-stamp space* C defined by the constraints

$$\varphi_C : \bigwedge_{j \in K} a_j \leq t_j - t_{j-1} \leq b_j$$

and computing the volume of any subset $C' \subseteq C$ amounts to computing the volume of its T' image $D' \subseteq D$. Let us remark that the density of t_j is the *convolution* of the densities $\psi_1, \ldots, \psi_j$ and its support is the Minkowski sum of $I_1, \ldots, I_j$.

The time-stamp space C and its subsets that we will encounter are defined as conjunctions of inequalities of the form $x \prec c$ or $x - x' \prec c$ where $\prec \in \{<, \leq, =, \geq, >\}$ and c is an integer constant. They define polytopes which are called *zones* (or timed polyhedra). Zones are used extensively in the analysis of timed automata [23,17,26]. They admit an efficient representation by *difference-bound matrices* (DBM) [19] and efficient algorithms based on shortest-path to remove redundant constraints [16].

Definition 3 (Process System). *A process system consists of n processes*

$$S = P^{\mathbf{1}} || \cdots || P^{\mathbf{n}} = \{(\mathcal{I}^{\mathbf{i}}, \Psi^{\mathbf{i}})\}_{i=1}^{n}$$

We use notations $P_j^{\mathbf{i}}$ to refer to step j of process i and $I_j^{\mathbf{i}} = [a_j^{\mathbf{i}}, b_j^{\mathbf{i}}]$ and $\psi_j^{\mathbf{i}}$ for the respective intervals and densities. To ease notation we assume all processes to have the same number k of steps. The event alphabet of the system is

$$\Sigma = \{e_1^{\mathbf{1}}, e_2^{\mathbf{1}}, \ldots, e_{k-1}^{\mathbf{n}}, e_k^{\mathbf{n}}\}$$

consisting of all the termination events of the steps of the various processes.

[3] These are the *timed traces* used originally in [4] to give semantics to timed automata. More about the relation between semantic models of timed behaviors can be found in [5].

A behavior of the system is induced by a point in the global duration space

$$y = (y_1^{\mathbf{1}}, y_2^{\mathbf{1}}, \ldots, y_{k-1}^{\mathbf{n}}, y_k^{\mathbf{n}}) \in \mathcal{D} = \prod_{i=1}^{n} \prod_{j=1}^{k} I_j^{\mathbf{i}} \subset \mathbb{R}^{nk},$$

which can be transformed into a point t in the time-stamp space

$$t = (t_1^{\mathbf{1}}, t_2^{\mathbf{1}}, \ldots, t_{k-1}^{\mathbf{n}}, t_k^{\mathbf{n}}) \in \mathcal{C} = T\mathcal{D}$$

where T is the appropriate block diagonal matrix.

When all processes start simultaneously, the time stamps are taken from the same global time reference and one can view a global run as merging local runs and sorting the events according to their time stamps, as illustrated in Fig. 2. The set of all such global behaviors is denoted by

$$L = L^{\mathbf{1}} || \cdots || L^{\mathbf{n}}.$$

All timed behaviors that admit the same *order* of events are said to exhibit the same *qualitative behavior*. This can be formalized as an operation among the untimed local languages. Let $\underline{L}^{\mathbf{i}} = \{e_1^{\mathbf{i}}\, e_2^{\mathbf{i}} \cdots e_k^{\mathbf{i}}\}$ be the untimed language associated with process $P^{\mathbf{i}}$: it consists of the unique qualitative behavior which satisfies the precedence constraints of $P^{\mathbf{i}}$. The potential qualitative behaviors of S constitute the language

$$\underline{L} = \underline{L}^1 || \cdots || \underline{L}^n$$

which is the *shuffle* of these languages, that is, the set of sequences consisting of one occurrence of each event in Σ and respecting the local precedence constraints for each process. Mathematically speaking, a qualitative behavior corresponds to a linear order[4] which is consistent with the partial order defined by the union of the precedence relations of all the tasks. Such an order is also known as *interleaving* in the theory of *concurrency* (motivated readers might want to consult [18] or [20]).

We use the term qualitative behavior also for any *prefix* of a sequence in $\underline{L}$. Such a prefix corresponds naturally to an *incomplete* run where not all processes have finished all their steps. From the standpoint of automata, qualitative behaviors correspond to *paths* in the transition graph of the *global automaton* associated with the system which is the (Cartesian) product $\mathcal{A} = \mathcal{A}^{\mathbf{1}} || \cdots || \mathcal{A}^{\mathbf{n}}$ of the automata associated with the individual processes as illustrated in Fig. 3. Unfortunately, these extremely important objects are not easy to draw for non-trivial dimensions. Incomplete behaviors correspond to paths not reaching the final state.

In a global *state* of the form $(q_{j_1}^{\mathbf{1}}, \ldots, q_{j_n}^{\mathbf{n}})$ each process i is busy executing its step j_i and there is a *race* between the termination transitions. The transition $e_{j_i}^{\mathbf{i}}$ that will win will be the first to satisfy the condition $x^{\mathbf{i}} = y_{j_i}^{\mathbf{i}}$. Since $x^{\mathbf{i}}$ has been reset to zero at $t_{j_i-1}^{\mathbf{i}}$ this condition will be fulfilled at time $t_{j_i-1}^{\mathbf{i}} + y_{j_i}^{\mathbf{i}} = t_{j_i}^{\mathbf{i}}$. The outcomes of all these races are completely determined by the value of y, and this determines the qualitative behavior which is exhibited. Had there been no timing constraints on task

[4] Since we are dealing with volumes, our neglect of the possibility of events occurring at *exactly* the same time and not paying too much attention to the distinction between strict and non strict inequalities is justified.

durations, that is, $I^i_j = [0, \infty)$, the system would be completely *asynchronous* and all interleavings would, in principle, be possible. When durations are bounded, some qualitative behaviors may become strictly impossible due to the arithmetics of timing constraints while others will occur at low probability. In the sequel we develop methods for computing these probabilities.

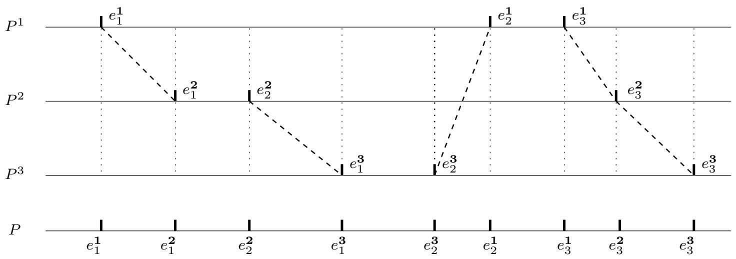


Fig. 2. A global behavior $w = e^1_1\ e^2_1\ e^2_2\ e^3_1\ e^2_3\ e^1_2\ e^1_3\ e^3_2\ e^3_3$ obtained by merging local behaviors. The dashed line indicate the minimal set of additional inter-process constraints that characterize w.

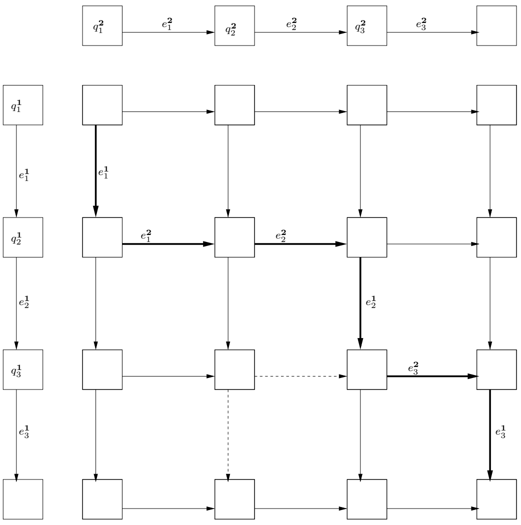


Fig. 3. The product automaton for a process system with $n = 2$, $k = 3$. The thick arrows indicate the path corresponding to the qualitative behavior $w = e^1_1\ e^2_1\ e^2_2\ e^3_1\ e^2_3\ e^1_2\ e^1_3\ e^3_2\ e^3_3$. The race between e^1_3 and e^2_2 in state (q^1_3, q^2_2) is indicated by the dashed arrows.

3 Computing Volumes

The computation of the probability of a qualitative behavior w is performed in two steps. First we associate with it a zone $Z_w \subseteq \mathcal{C}$ consisting of all instances of t that yield this behavior. Then we integrate over this zone to find its volume.

Let φ_C be the constraint describing the whole time-stamp space:

$$\varphi_C : \bigwedge_{i \in N} \bigwedge_{j \in K} a_j^{\mathbf{i}} \leq t_j^{\mathbf{i}} - t_{j-1}^{\mathbf{i}} \leq b_j^{\mathbf{i}}$$

with $t_0^{\mathbf{i}} = 0$ for every i. The zone Z_w for the qualitative behavior of Fig. 2 can be characterized by adding constraints that specify the particular order of events in w:

$$\varphi_w : \varphi_C \wedge\ t_1^{\mathbf{1}} < t_1^{\mathbf{2}} < t_2^{\mathbf{2}} < t_1^{\mathbf{3}} < t_3^{\mathbf{2}} < t_2^{\mathbf{1}} < t_3^{\mathbf{1}} < t_2^{\mathbf{3}} < t_3^{\mathbf{3}}.$$

Some of these constraints appear already in φ_C and some are implied via transitivity by other constraints. After eliminating these redundant constraints one obtains the following description:

$$\varphi_w : \varphi_C \wedge (t_1^{\mathbf{1}} < t_1^{\mathbf{2}}) \wedge (t_2^{\mathbf{2}} < t_1^{\mathbf{3}}) \wedge (t_2^{\mathbf{3}} < t_2^{\mathbf{1}}) \wedge (t_3^{\mathbf{1}} < t_3^{\mathbf{2}}) \wedge (t_3^{\mathbf{2}} < t_3^{\mathbf{3}}).$$

As illustrated in Fig. 2, the constraints that remain in φ_w are the inter-process constraints that are sufficient to characterize w. These constraints can be computed *incrementally* as we move along the prefix of a qualitative behavior. Let us follow the first two steps. Initially we have the empty word whose associated zone is $\mathcal{C}$ and hence its probability is 1. After the occurrence of the first event $e_1^{\mathbf{1}}$ we know that $P_1^{\mathbf{1}}$ terminated before $P_1^{\mathbf{2}}$ and $P_1^{\mathbf{3}}$. This leads to the constraints:

$$\varphi_{e_1^{\mathbf{1}}} : \varphi_C \wedge (t_1^{\mathbf{1}} < t_1^{\mathbf{2}})\ \wedge\ (t_1^{\mathbf{1}} < t_1^{\mathbf{3}}) \tag{2}$$

After this first event we have a competition between $e_1^{\mathbf{2}}$, $e_1^{\mathbf{3}}$ and $e_2^{\mathbf{1}}$. The winner of the race is the next event of w, $e_1^{\mathbf{2}}$ and hence we add the constrains $t_1^{\mathbf{2}} < t_1^{\mathbf{3}}$ and $t_1^{\mathbf{2}} < t_2^{\mathbf{1}}$ and remove the constraint $t_1^{\mathbf{1}} < t_1^{\mathbf{3}}$ which becomes redundant, yielding:

$$\varphi_{e_1^{\mathbf{1}} e_1^{\mathbf{2}}} : \varphi_C \wedge (t_1^{\mathbf{1}} < t_1^{\mathbf{2}})\ \wedge\ (t_1^{\mathbf{2}} < t_1^{\mathbf{3}})\ \wedge\ (t_1^{\mathbf{2}} < t_2^{\mathbf{1}}).$$

In general whenever event $e_j^{\mathbf{i}}$ occurs, we add a constraint stating that $t_j^{\mathbf{i}}$ is smaller than the time stamps associated with all the pending events in the other processes. The incremental process is illustrated in Fig. 4.

This procedure is probabilistically correct in the following sense. For every $w \in \underline{L}$ the probability of all behaviors having w as a prefix is the relative volume of the corresponding zone Z_w, namely, $p(w) = |Z_w|/|\mathcal{C}|$. This holds trivially for the empty behavior when there are no constraints. For the inductive step observe that any qualitative behavior of the form $w\ e$ which extends w has to satisfy φ_w due to *causality* as well as additional constraints that guarantee that e is indeed the next event to win the race. The constraints associated with all the extension of w form a *partition* of Z_w and all the probabilistic mass $p(w)$ is split among them, satisfying

$$\sum_e p(w\ e) = p(w).$$

In [29] a similar incremental approach that goes from a path/prefix to its successors has been developed using the clock auxiliary variables. The use of clocks required the

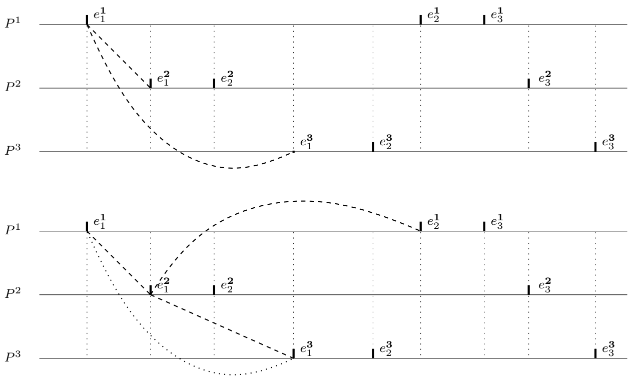


Fig. 4. Incremental constraint construction: constraints for $e^{\mathbf{1}}_1$ and then for $e^{\mathbf{1}}_1\ e^{\mathbf{2}}_1$. The constraint $t^{\mathbf{1}}_1 < t^{\mathbf{3}}_1$ becomes redundant after the second event.

concept of *density transformers* to account for the distribution of clock values before and after transitions (see also [2,10,14,30]). These are not needed in the clock-free approach presented here. Those acquainted with the verification of timed automata using a forward computation of the simulation/reachability graph [17,26] may notice that for every w the zone Z_w in the time-stamp space is empty exactly when its associated clock space zone in the reachability graph becomes empty. This suggests an alternative clock-free analysis algorithm for timed automata which is immediately applicable to acyclic systems but will require more work to be adapted to the cyclic case.

Having labeled qualitative behaviors by constraints we need to compute the volume of the zones. We illustrate this procedure on a concrete example with $n = 3$ and $k = 1$, hence $\mathcal{D} = \mathcal{C} = I^{\mathbf{1}}_1 \times I^{\mathbf{2}}_1 \times I^{\mathbf{3}}_1$, with concrete values

$$[a^{\mathbf{1}}_1, b^{\mathbf{1}}_1] = [2, 5],\ \ [a^{\mathbf{2}}_1, b^{\mathbf{2}}_1] = [3, 4],\ \text{ and }\ [a^{\mathbf{3}}_1, b^{\mathbf{3}}_1] = [4, 7].$$

The constraints associated with all qualitative behaviors where process $P^{\mathbf{1}}$ wins the first race are

$$\varphi_{e^{\mathbf{1}}_1} :\ \ (2 \le t^{\mathbf{1}}_1 \le 5) \wedge (3 \le t^{\mathbf{2}}_1 \le 4) \wedge (4 \le t^{\mathbf{3}}_1 \le 7) \wedge (t^{\mathbf{1}}_1 < t^{\mathbf{2}}_1) \wedge (t^{\mathbf{1}}_1 < t^{\mathbf{3}}_1).$$

We pick an integration order $t^{\mathbf{3}}_1 \prec t^{\mathbf{2}}_1 \prec t^{\mathbf{1}}_1$, that is, the inside-out order of variable elimination, and rewrite $\varphi_{e^{\mathbf{1}}_1}$ as

$$\varphi_{e^{\mathbf{1}}_1} :\ \ (2 \le t^{\mathbf{1}}_1 \le 5) \wedge (\max(3, t^{\mathbf{1}}_1) \le t^{\mathbf{2}}_1 \le 4) \wedge (\max(4, t^{\mathbf{1}}_1) \le t^{\mathbf{3}}_1 \le 7)$$

Then we split $I^{\mathbf{1}}_1$ into maximal segments where both $\max(3, t^{\mathbf{1}}_1)$ and $\max(4, t^{\mathbf{1}}_1)$ are uniform. In our example $[2, 5]$ splits into $[2, 3]$, $[3, 4]$ and $[4, 5]$ and the volume of the set can be written as

$$\left[\int_2^3 \int_3^4 \int_4^7 + \int_3^4 \int_{t^{\mathbf{1}}_1}^4 \int_4^7 + \int_4^5 \int_{t^{\mathbf{1}}_1}^4 \int_{t^{\mathbf{1}}_1}^7\right] dt^{\mathbf{3}}_1 dt^{\mathbf{2}}_1 dt^{\mathbf{1}}_1 = 3 + \frac{3}{2} + 0 = \frac{9}{2}$$

which after dividing by $|\mathcal{C}| = 9$ gives a probability of $1/2$ for $e_1^{\mathbf{1}}$ winning the first race. Figure 5 illustrates two possible splits of a 2-dimensional zone into integration domains. The number of case splits and the forms of the integration domains may vary a lot depending on the chosen order.

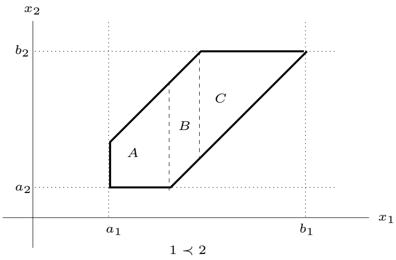


Fig. 5. Zone volume computation by splitting into integration domains in two different integration orders which yield 3 and 2 domains, respectively

Theorem 1 (Probability of Qualitative Behaviors). *Given a system of stochastic sequential processes as in Def. 3 the probability of any of its qualitative behaviors is computable.*

The global termination time (makespan in the job-shop jargon) of a behavior is $\Theta = \max\{t_k^{\mathbf{1}}, \ldots, t_k^{\mathbf{n}}\}$. For all behaviors that are qualitatively equivalent the maximum is attained by the same variable, namely $t_k^{\mathbf{i}}$ for any behavior whose last event is $e_k^{\mathbf{i}}$. To compute the expected termination time we integrate $t_k^{\mathbf{i}}$ over Z_w and sum up over all w:

$$\mathbb{E}(\Theta) = \frac{1}{|\mathcal{C}|} \sum_{i=1}^{n} \sum_{w=w'e_k^{\mathbf{i}}} \int_{Z_w} t_k^{\mathbf{i}}.$$

Before moving to the coordinated execution framework let us mention some useful observations. So far we have treated qualitative behaviors in their finest granularity, taking note of the ordering between any pair of events. In many situations we are interested in *sets* of qualitative behaviors and their probability can often be computed more efficiently than summing up the probabilities of individual qualitative behaviors.

Suppose we want to characterize the set of all qualitative behaviors that pass through a global state $q = (q_{j_1}^{\mathbf{1}}, \ldots, q_{j_n}^{\mathbf{n}})$. Let $\underline{L}_j^{\mathbf{i}} = \{e_1^{\mathbf{i}} \cdots e_{j-1}^{\mathbf{i}}\}$ be the qualitative behavior of $P^{\mathbf{i}}$ that leads to $q_j^{\mathbf{i}}$. Then the set of qualitative behaviors that lead to q is

$$\underline{L}(q) = \underline{L}_{j_1}^{\mathbf{1}} || \cdots || \underline{L}_{j_n}^{\mathbf{n}}.$$

The constraints that characterize $\underline{L}(q)$ may forget the specific interleaving, that is, the specific order in which *past* events have occurred. The only constraints that are relevant are those that guarantee that the entrance of each process into its respective local state preceded the exit of all other processes from their respective states, that is,

$$\varphi_q : \varphi_{\mathcal{C}} \wedge \bigwedge_{i=1}^{n} \bigwedge_{i' \neq i} t_{j_i - 1}^{\mathbf{i}} < t_{j_{i'}}^{\mathbf{i}'}.$$

Thus, to compute the expected termination time it suffices to partition the set of qualitative behaviors into n classes according to the identity of the *last* transition, letting Z^i be the zone defined by

$$\varphi^i : \varphi_{\mathcal{C}} \wedge \bigwedge_{i' \neq i} t_k^{\mathbf{i}'} < t_k^{\mathbf{i}}.$$

Then the expected termination time is

$$\mathbb{E}(\Theta) = \frac{1}{|\mathcal{C}|} \sum_{i=1}^{n} \int_{Z^i} t_k^{\mathbf{i}}. \tag{3}$$

A similar observation, made in the context of zone-based verification of timed automata, underlies the fact that the union of zones reached by interleavings of the same set of events is convex [31,27,9].

4 Conflicts and Schedulers

Now we adapt the framework to the case where steps of different processes may be at conflict due to requiring the same resource and hence cannot be executed simultaneously. Naturally, this situation is more intuitively expressed using automata, states and runs. In order not to discourage semantically challenged audiences we explain the automaton-based modeling very informally. Interested readers may consult [1] for a general framework for modeling and solving scheduling problems with timed automata as well as [29] for the formal definitions of *duration-probabilistic automata* (DPA) which is the model underlying this paper.

As a running example consider a system of two processes with three steps each, admitting a resource conflict between their respective second steps $P_2^{\mathbf{1}}$ and $P_2^{\mathbf{2}}$. Conflicts are modeled in automata using *forbidden states* in the global automaton, state $(q_2^{\mathbf{1}}, q_2^{\mathbf{2}})$ in our example. To be able to prevent the automaton from entering this state[5] we refine the process model so that the initiation of step $P_j^{\mathbf{i}}$ does not occur *automatically* upon the termination of step $P_{j-1}^{\mathbf{i}}$. We thus modify the process automaton shown in Fig. 3 by inserting a *waiting state* $\bar{q}_j^{\mathbf{i}}$ between $q_{j-1}^{\mathbf{i}}$ and $q_j^{\mathbf{i}}$. The automaton can leave this state only when it receives a *start* command $s_j^{\mathbf{i}}$ from a scheduler as illustrated in Fig. 6-(a).

As long as the scheduler is not completely specified the system is *open* or using another terminology, admits both *probabilistic* and *set-theoretic* non-determinism. For example in state $(\bar{q}_2^{\mathbf{1}}, q_1^{\mathbf{2}})$ process $P^{\mathbf{1}}$ may either start its second step $(\bar{q}_2^{\mathbf{1}}, q_1^{\mathbf{2}}) \rightarrow (q_2^{\mathbf{1}}, q_1^{\mathbf{2}})$ or wait until step $P_1^{\mathbf{2}}$ terminates and let $P^{\mathbf{2}}$ take the resource first $(\bar{q}_2^{\mathbf{1}}, q_1^{\mathbf{2}}) \rightarrow (\bar{q}_2^{\mathbf{1}}, \bar{q}_2^{\mathbf{2}}) \rightarrow (\bar{q}_2^{\mathbf{1}}, q_2^{\mathbf{2}})$. A scheduler resolves this type of non-determinism by telling each process in a waiting state whether to take the resource and proceed to execution or wait until the resource is taken and released by another process.[6] Once such a scheduler is defined, the

[5] We consider schedulers that by construction cannot make the system enter a forbidden state.

[6] Note that we restrict ourselves to *non-lazy* schedulers: if they do not issue an $s_j^{\mathbf{i}}$ command at some point, they will not issue it later unless another process has utilized the resource. This class has been shown [1] to contain the optimal schedulers for the problems we are dealing with.

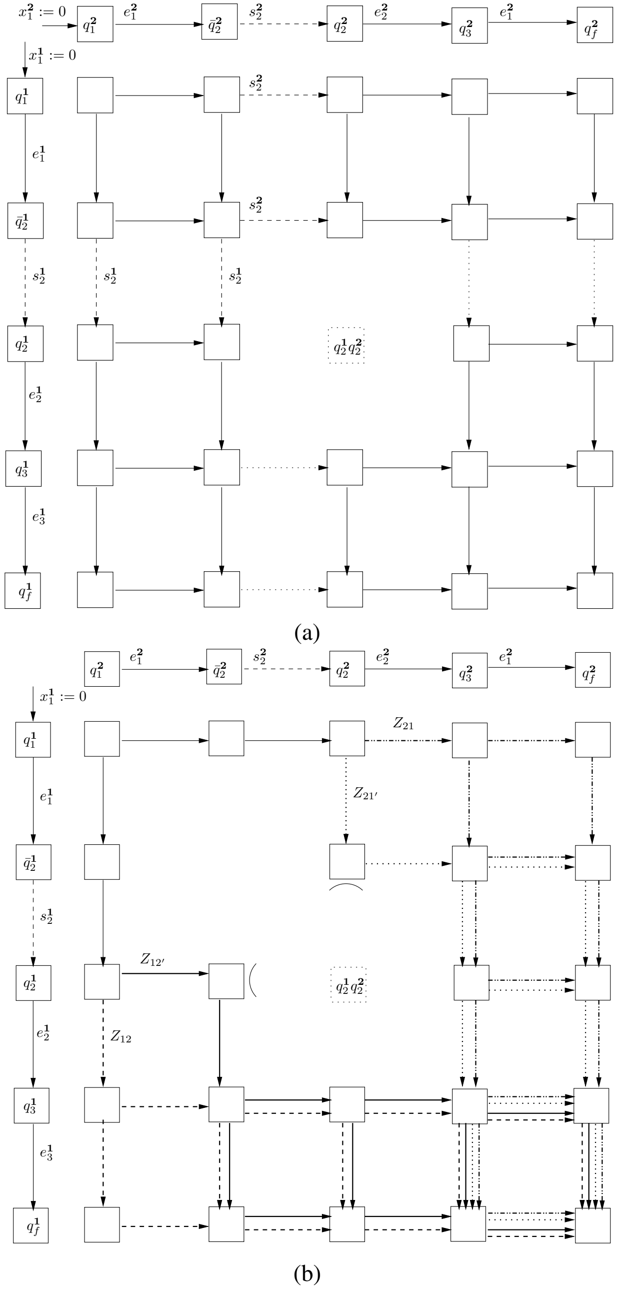


Fig. 6. (a) Two parallel processes admitting a resource conflict and their product automaton. The dashed arrows indicate *start* transitions which should be under the control of a scheduler while the dotted arrows indicate post-conflict *start* transitions; (b) The automaton resulting from composition with a FIFO scheduler and the 4 potential conflict resolution and resource utilization scenarios.

set-theoretic non-determinism is eliminated and the only non-determinism that remains is the one associated with task durations and thus it becomes possible to compute probabilities. To be more precise, probabilities can be computed also for non-deterministic schedulers that make a probabilistic choice, but we do not consider them here.

A scheduler is thus a mechanism which may observe the state of the system and decide whether to grant a resource to a process, possibly based on the level of progress of other processes. The most passive scheduler grants the resource to the *first* process whose corresponding step becomes enabled. Under such a FIFO scheduling policy it is the result of the race between $e_1^{\mathbf{1}}$ and $e_1^{\mathbf{2}}$ which determines the resource granting decision. The automaton obtained by composing the system with such a scheduler is shown in Figure 6-(b) where we have chosen to ignore the zero-measure situation when both processes terminate *exactly* at the same time (alternatively this situation can be handled by assigning an arbitrary priority when this is the case).

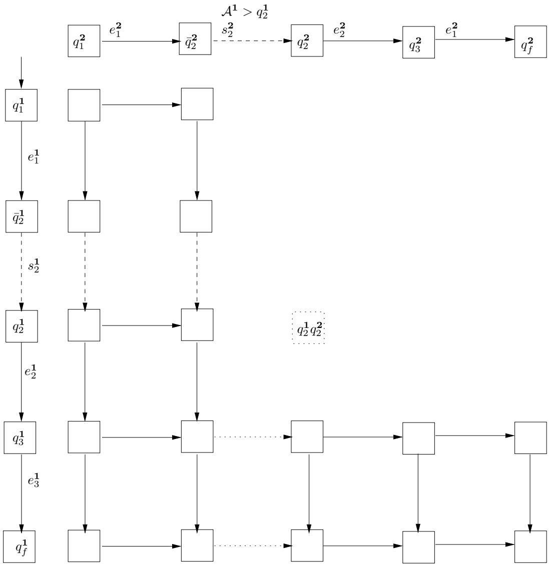


Fig. 7. (a) A scheduler that gives strict priority to $P^{\mathbf{1}}$. This is realized by the condition $\mathcal{A}^{\mathbf{1}} > q_2^{\mathbf{1}}$ which allows $P_2^{\mathbf{2}}$ to start only after $P_2^{\mathbf{1}}$ terminates.

More active schedulers interfere with the execution order by imposing additional conditions upon the *start* transitions. Suppose that the duration of step $P_3^{\mathbf{1}}$ is much longer than that of $P_3^{\mathbf{2}}$ hence it would be reasonable to give $P_2^{\mathbf{1}}$ a priority over $P_2^{\mathbf{2}}$ even if the latter becomes enabled earlier. This priority can have different degrees of rigidity. A *strict* priority scheduler allows $s_2^{\mathbf{2}}$ only in global states where $P_2^{\mathbf{1}}$ has terminated, a condition that we write as $\mathcal{A}^{\mathbf{1}} > q_2^{\mathbf{1}}$. The automaton obtained by composing the system with such a scheduler is shown in Fig. 7. Note that strict priority schedulers make the automaton always "bypass" a conflict state from the same side.

Strict priority schedulers can be unnecessarily rigid for tasks with durations variability as they do not adapt to the actual evolution of the schedule. As an example for such adaptability consider the case where $P_1^{\mathbf{2}}$ terminates very early so that we can start $P_2^{\mathbf{2}}$ so that it will surely terminate before $P_2^{\mathbf{1}}$ becomes enabled and hence will not block it. Even if this is not guaranteed with certainty, a scheduler might want to start $P_2^{\mathbf{2}}$ if the expected delay incurred to $P^{\mathbf{1}}$ is small. Technically, the knowledge of the relative timing of $e_1^{\mathbf{2}}$ at decision time is encoded by the value of clock $x^{\mathbf{1}}$ reset upon starting $P_1^{\mathbf{1}}$. The larger is the value of $x^{\mathbf{1}}$, the more we are likely to block $P^{\mathbf{1}}$ and for a longer period. Hence the condition for issuing $s_2^{\mathbf{2}}$ by such a state-dependent scheduler will be of the form $(\mathcal{A}^{\mathbf{1}} > q_2^1) \vee (\mathcal{A}^{\mathbf{1}} < \bar{q}_2^1 \wedge x^{\mathbf{1}} < d)$ for some constant d.

The labeling of states and qualitative behaviors with constraints in order to compute volumes, probabilities and expected termination times can be extended to handle all these types of schedulers. As an illustration consider the FIFO scheduler of Fig. 6-(b) which admits 4 classes of qualitative behaviors (scenarios) that correspond to the outcomes of the conflict between $P^{\mathbf{1}}$ and $P^{\mathbf{2}}$ on the shared resource. These scenarios are characterized by the identity of the winner (for this scheduler it depends on the relation between $t_1^{\mathbf{1}}$ and $t_1^{\mathbf{2}}$) and by whether the loser termination time is delayed (depending on whether the winner releases the resource before the loser becomes enabled). These cases are summarized in Table 1 and depicted in Fig. 6-(b).

Table 1. The zones corresponding to the four possible outcomes of the resource conflict of Fig. 7-(b). Constraints on $t_3^{\mathbf{1}}$ and $t_3^{\mathbf{2}}$ are omitted.

winner	loser delayed		loser not delayed
	$Z_{12'}$		Z_{12}
$P^{\mathbf{1}}$	$a_1^{\mathbf{1}} < t_1^{\mathbf{1}} < b_1^{\mathbf{1}}$		$a_1^{\mathbf{1}} < t_1^{\mathbf{1}} < b_1^{\mathbf{1}}$
	$a_1^{\mathbf{2}} < t_1^{\mathbf{2}} < b_1^{\mathbf{2}}$		$a_1^{\mathbf{2}} < t_1^{\mathbf{2}} < b_1^{\mathbf{2}}$
	$t_1^{\mathbf{1}} < t_1^{\mathbf{2}}$		$t_1^{\mathbf{1}} < t_1^{\mathbf{2}}$
	$t_1^{\mathbf{1}} + a_2^{\mathbf{1}} < t_2^{\mathbf{1}} < t_1^{\mathbf{1}} + b_2^{\mathbf{1}}$		$t_1^{\mathbf{1}} + a_2^{\mathbf{1}} < t_2^{\mathbf{1}} < t_1^{\mathbf{1}} + b_2^{\mathbf{1}}$
	$t_1^{\mathbf{2}} < t_2^{\mathbf{1}}$		$t_2^{\mathbf{1}} < t_1^{\mathbf{2}}$
	$t_2^{\mathbf{1}} + a_2^{\mathbf{2}} < t_2^{\mathbf{2}} < t_2^{\mathbf{1}} + b_2^{\mathbf{2}}$		$t_1^{\mathbf{2}} + a_2^{\mathbf{2}} < t_2^{\mathbf{2}} < t_1^{\mathbf{2}} + b_2^{\mathbf{2}}$
	$Z_{21'}$		Z_{21}
$P^{\mathbf{2}}$	$a_1^{\mathbf{1}} < t_1^{\mathbf{1}} < b_1^{\mathbf{1}}$		$a_1^{\mathbf{1}} < t_1^{\mathbf{1}} < b_1^{\mathbf{1}}$
	$a_1^{\mathbf{2}} < t_1^{\mathbf{2}} < b_1^{\mathbf{2}}$		$a_1^{\mathbf{2}} < t_1^{\mathbf{2}} < b_1^{\mathbf{2}}$
	$t_1^{\mathbf{2}} < t_2^{\mathbf{1}}$		$t_1^{\mathbf{2}} < t_2^{\mathbf{1}}$
	$t_1^{\mathbf{2}} + a_2^{\mathbf{2}} < t_2^{\mathbf{2}} < t_1^{\mathbf{2}} + b_2^{\mathbf{2}}$		$t_1^{\mathbf{2}} + a_2^{\mathbf{2}} < t_2^{\mathbf{2}} < t_1^{\mathbf{2}} + b_2^{\mathbf{2}}$
	$t_2^{\mathbf{2}} < t_1^{\mathbf{1}}$		$t_1^{\mathbf{1}} < t_2^{\mathbf{2}}$
	$t_1^{\mathbf{2}} + a_2^{\mathbf{2}} < t_2^{\mathbf{2}} < t_1^{\mathbf{2}} + b_2^{\mathbf{2}}$		$t_1^{\mathbf{2}} + a_2^{\mathbf{2}} < t_2^{\mathbf{2}} < t_1^{\mathbf{2}} + b_2^{\mathbf{2}}$

As the alert reader might have noticed, the transformation T from the duration space to the time-stamp space is different from the independent execution framework. It can nevertheless be shown to be volume preserving along the following lines. First, one can show that after adding inter-process precedence constraints causality is preserved and there is always a rearrangement of the indices such that the transformation matrix remains lower triangular. Secondly the notion of volume preservation can be easily generalized from linear to piecewise-linear transformations.

The above analysis can be generalized to m distinct resources and to multi-party conflicts on each of them. For each resource l one can compute the set U_l of all the utilization scenarios for this resource and their respective zones. A scenario corresponds to a particular order of resource utilization by conflicting steps and to the waiting delays incurred to these steps. Then the classes of potential qualitative behaviors of interest are the combinations of those, that is, $U = U_1 \times \cdots \times U_m$ with zones defined by intersection. While this sounds like a recipe for a severe combinatorial explosion, note that many scenarios will lead to empty zones, either for logical reasons (inter-process ordering of conflicting steps is incompatible with local precedence constraints) or due to the arithmetics of timing constraints (two conflicting tasks, one at the beginning and one at the end of their respective processes, are likely to be executed in one order). Naturally, for priority schedulers there will be fewer scenarios to analyze.

We have implemented a prototype tool which computes expected termination times as described in this paper. As input it takes a system description consisting of processes, steps, duration intervals and conflicts as well as a definition of a scheduling policy. Then for every utilization scenario it derives the corresponding zone, using the DBM library of IF [12] to normalize constraints and detect empty zones. Then it performs integration over the non-empty zones to compute probability and expected termination time. The integration uses the *GNU Multiple Precision Arithmetic Library* (GMP) to avoid rounding errors.

Let us describe our preliminary experiments. For each value of n from 1 to 5 and for each value of k from 1 to 40, we choose a number of conflicts (between 0 and 3) and a number of participants in each conflict (2 or 3). Each choice in this space defines a problem type for which we draw 10 concrete problems by randomly choosing the identity of the conflicting steps as well as step duration intervals of the form $[c-d, c+d]$ with c drawn uniformly in $[40, 50]$ and d in $[0, c/20]$. Then we try to compute expected termination times for a FIFO scheduler with a timeout of 3 minutes per problem on an old laptop.

The experiments with $n = 1$ compute the volume of one zone, the time-stamp space. Applying the reverse order integration we can compute up to dimension 63 in 0.4 seconds (currently this is a limitation of our DBM library). In general integration takes place in $\mathbb{R}^{nk}$ and its complexity depends on the following factors. First, the number of scenarios (orders of resource utilizations and their combinations) determines the number of zones whose volume we might need to compute, in case they are not detected beforehand to be empty. For each zone and each variable t we compute I_t, the projection of the zone on t. Then we define a partial order relation between these intervals such that $I_t < I_{t'}$ if the upper bound of I_t is smaller than the lower bound of $I_{t'}$. Then we construct a compatible linear order and integrate backwards. The chosen order determines the number of case splits but also the form of the integration domains and the polynomials obtained during integration. We experienced orders of integration that generate more splits but take less overall computation time. Since there is a lot of exploration and fine tuning ahead it is premature to report performance systematically. To give an idea, we mention some problem types for which we managed to compute for all the test cases.These include $(n, k) = (2, 12)$ with 2 conflicts, $(3, 6)$ and $(4, 6)$ with 3 binary conflicts or 2 ternary conflicts and $(5, 4)$ with 2 binary conflicts.

5 Concluding Remarks

We have presented a computational technique to evaluate schedulers in a non-Markovian setting based on splitting the space of valuations of the random variables and computing volumes. To the best of our knowledge no similar computational results have been reported. We mention some future work.

1. Integration over zones is the major computational activity in our procedure and its optimization is an interesting algorithmic problem.
2. To handle larger systems one needs to develop algorithms that do not explore all classes of qualitative behaviors but restrict the exploration to a high-volume small subset of those, whenever such exists.
3. While this work solves the *analysis* problem it would be more challenging to *synthesize* optimal schedulers automatically using value iteration. It is an open question whether such a backward iteration can be defined using the clock-free methods developed in this paper.
4. Another major challenge is to extend this framework to cyclic systems, define the appropriate performance measures and study their steady-state behavior.
5. Finally, it would be interesting to compare the analytic method developed here with statistical approaches based on random simulation. It is intriguing to see how many simulation runs are needed to approximate our results with a good confidence.

Acknowledgment. This work grew out of numerous discussions with Kim Larsen and Bruce Krogh and benefitted from the constructive criticism of Eugene Asarin.

References

1. Abdeddaïm, Y., Asarin, E., Maler, O.: Scheduling with timed automata. Theoretical Computer Science 354(2), 272–300 (2006)
2. Alur, R., Bernadsky, M.: Bounded model checking for GSMP models of stochastic real-time systems. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 19–33. Springer, Heidelberg (2006)
3. Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking for probabilistic real-time systems. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) ICALP 1991. LNCS, vol. 510, pp. 115–126. Springer, Heidelberg (1991)
4. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
5. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. J. ACM 49(2), 172–206 (2002)
6. Asarin, E., Maler, O.: As soon as possible: Time optimal control for timed automata. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, pp. 19–30. Springer, Heidelberg (1999)
7. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: Proc. IFAC Symposium on System Structure and Control, pp. 469–474 (1998)
8. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Software Eng. 29(6), 524–541 (2003)
9. Ben-Salah, R., Bozga, M., Maler, O.: On interleaving in timed automata. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 465–476. Springer, Heidelberg (2006)

10. Bernadsky, M., Alur, R.: Symbolic analysis for GSMP models with one stateful clock. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 90–103. Springer, Heidelberg (2007)
11. Bouyer, P.: From Qualitative to Quantitative Analysis of Timed Systems. Mémoire d'habilitation, Université Paris (July 2009)
12. Bozga, M., Graf, S., Mounier, L.: IF-2.0: A validation environment for component-based real-time systems. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 343–348. Springer, Heidelberg (2002)
13. Brinksma, E., Hermanns, H., Katoen, J.-P. (eds.): EEF School 2000 and FMPA 2000. LNCS, vol. 2090. Springer, Heidelberg (2001)
14. Carnevali, L., Grassi, L., Vicario, E.: State-density functions over DBM domains in the analysis of non-Markovian models. IEEE Trans. Software Eng. 35(2), 178–194 (2009)
15. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems, 2nd edn. Springer, Heidelberg (2008)
16. Cormen, T.T., Leiserson, C.E., Rivest, R.L.: Introduction to algorithms. MIT Press, Cambridge (1990)
17. Daws, C., Olivero, A., Tripakis, S., Yovine, S.: The tool KRONOS. In: Hybrid Systems, pp. 208–219 (1995)
18. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scientific, Singapore (1995)
19. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Automatic Verification Methods for Finite State Systems, pp. 197–212 (1989)
20. Esparza, J., Heljanko, K.: Unfoldings – A Partial-Order Approach to Model Checking. Springer, Heidelberg (2008)
21. German, R.: Non-markovian analysis. In: Brinksma, E., et al. (eds.) [13], pp. 156–182.
22. Glynn, P.W.: A GSMP formalism for discrete event systems. Proceedings of the IEEE 77(1), 14–23 (1989)
23. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Information and Computation 111(2), 193–244 (1994)
24. Jurdzinski, M., Peled, D., Qu, H.: Calculating probabilities of real-time test cases. In: Grieskamp, W., Weise, C. (eds.) FATES 2005. LNCS, vol. 3997, pp. 134–151. Springer, Heidelberg (2006)
25. Kartson, D., Balbo, G., Donatelli, S., Franceschinis, G., Conte, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley & Sons, Chichester (1994)
26. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer (STTT) 1(1), 134–152 (1997)
27. Lugiez, D., Niebert, P., Zennou, S.: A partial order semantics approach to the clock explosion problem of timed automata. Theoretical Computer Science 345, 27–59 (2005)
28. Maler, O.: On optimal and reasonable control in the presence of adversaries. Annual Reviews in Control 31(1), 1–15 (2007)
29. Maler, O., Larsen, K.G., Krogh, B.H.: On zone-based analysis of duration probabilistic automata. In: INFINITY, pp. 33–46 (2010)
30. Vicario, E., Sassoli, L., Carnevali, L.: Using stochastic state classes in quantitative evaluation of dense-time reactive systems. IEEE Trans. Software Eng. 35(5), 703–719 (2009)
31. Zhao, J.: Partial order path technique for checking parallel timed automata. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 417–432. Springer, Heidelberg (2002)

Formal Modeling and Analysis of Timed Systems: Technology Push or Market Pull?

Boudewijn R. Haverkort[1,2]

[1] Embedded Systems Institute, Eindhoven
boudewijn.haverkort@esi.nl
http://www.esi.nl/

[2] Design and Analysis of Communication Systems, University of Twente
http://www.utwente.nl/ewi/dacs/

Abstract. In this short paper I will address the question whether the methods and techniques we develop are applied well in industrial practice. To address this question, I will make a few observations from the academic field, as well as from industrial practice. This will be followed by a concise analysis of the cause of the perceived gap between the academic state-of-the-art and industrial practice. I will conclude with some opportunities for improvement.

1 Background

Twenty-five years ago, in september 1986, I received an engineering degree in Computer Science from the University of Twente. Since then, I have been working in the field of performance and dependability evaluation of computer and communication systems, either developing theory, methods and tools, or working on applications. My work on theory includes Markovian models of all sorts, queueing network analysis techniques, rare event simulation techniques, quasi-birth-death processes (QBDs), and, over the last 10 years, much work on stochastic model checking [2].

Although I have published by far the most on theory and tools, cf. [10], I did address quite some application fields as well, including computer networking (e.g., ATM and B-ISDN networks, WLAN and token ring access networks, TCP/IP flow and congestion control), fault-tolerant distributed systems (e.g., fault-tolerant multiprocessor systems, system survivability with application to the Google file system), and embedded systems (e.g., embedded control systems, or wearable power-constrained communication devices). Lately, I have become more involved in SCADA systems, as well as with intrusion detection in network systems as part of my activities at the University of Twente, and with a wide variety of embedded computer-control systems as part of my activities at the Embedded Systems Institute (ESI), the latter in close cooperation with leading high-tech industries such as ASML, NXP, Océ, Philips and Thales.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 18–24, 2011.

2 Key Question Addressed

Being able to look back at 25 years of scientific work in this field, it is a good time to ask what has been achieved in the generic field of modeling and analysis of timed systems, and how valuable that has been, and for whom?

In the past three years, I already had the opportunity to reflect on the achievements of the field of performance evaluation at large, through invited presentations at three European conferences [5,8,9]. In this short paper, I would like to go a step further, and take my experience with industry while being employed at the ESI since 2009, much more into account. The key question that I address then is: ***Are the methods and techniques we develop being used in industrial practice?*** With ***we*** in this question, I do mean the academic community, that is, the typical participants of conferences like FORMATS, QEST, CAV, E-PEW, MASCOTS, etc.

To address this question, I will start with a few observations from the academic field, as well as from industrial practice, followed by a concise analysis, and some directions towards the future.

3 Observations from the Academic Field

Let me first address five observations that I think I can honestly make about the academic field that addresses models and techniques for the evaluation of quantitative system properties, such as timeliness, performance and reliability. I restrict to five only, for conciseness, the list of course is longer and more detailed. The list is not intended to blame anyone or any sub-community; I am just critically assessing the work being done by the community I have been working in happily myself for 25 years!

A1. The key theoretical results in most operations research related fields, such as queueing networks, stochastic Petri nets, optimization and discrete-event simulation, have been obtained 25 years or longer ago. The number of truly new results in this field in the last 25 years is rather small; one can think about models for self-similar traffic, improved EM-fitting of phase-type distributions, or the logarithmic reduction algorithm for QBDs.

A2. Important new results have been achieved in the field of explicit state-space-based methods, including storage structures like BDD's and MTBDD's, as well as efficient model checking and numerical algorithms, in the last 25 years.

A3. Key results have been attained regarding the structured (sometimes compositional) description of systems and system behavior, using model- or analysis-specific languages, having a formal basis, thus forming a good starting point for further analysis and synthesis. Think of timed-automata of various sorts, advances in process algebra, etc.

A4. Much more developer-friendly (this is not necessarily *user*-friendly) and powerful software tools have become available, that allow for larger case studies to be addressed with the techniques referred to in A1–3. Note that this trend is largely a result the availability of commodity computing power since the mid 1980's, hence, of Moore's law, and far less so of academic improvements.

A5. A large share of the recent work in academia is addressing (minor) extensions of methods and techniques (cf. A1–2), of description techniques (cf. A3) as well of as supporting tools (cf. A4), simply because theory allows us to do so ("technology push"), rather than that there is well-pronounced need for this from industry ("market pull").[1]

4 Observations and Trends in Industry

Similar as done for academia, let me also address five key observations from industry. Again, the list is longer, and there will be nuances to all of the items. I compiled this list as by-product of the process towards a new research agenda for ESI [4]. As such, it primarily addresses the embedded systems field, however, I do think that these observation apply more generally.

I1. There is an ever-increasing growth of complexity of systems and system software, for a variety of reasons, including the inclusion of legacy code or third-party components, increased openness, higher user expectations, the use of multicore hardware, mixed criticality of the software, and the advent of systems-of-systems.

I2. Given this increasing complexity, both system design and system test and integration, in terms of the classical V-model for system development, are severely challenged, technically, but also regarding time and cost constraints.

I3. There is an increased interest in model-driven design, however, in most industries with a focus on modeling as a kind of high-level programming, and code generation taking over part of the intensive coding work, thus saving time and costs.

I4. A sense of urgency regarding modeling and quantitative system evaluation is felt, but it is undirected as of yet. Modeling and analysis, as we advocate it, is not a natural component in the design process of software-intensive systems. This is very much unlike the practice in many other system design areas, like hardware, mechanical or optical system design.

I5. When modeling is used to guide the design process, that is, to explore the design space and to make trade-offs, this is either done on the basis of back-of-the-envelope models (or spread-sheets at best), or on the basis of extremely detailed simulation models that are, in fact, system prototypes implemented in a simulator environment. In most cases, the modeling techniques we have been developing, are *not* being used.

[1] A colleague of mine, from a different field in computer science, referred to this as "Harley Davidson extension work", liked by Harley Davidson hobbyists, irrelevant for anyone else, but annually displayed at length at nic(h)e workshops and conferences around the world.

5 Analysis

As should be clear from the two sets of observations, there is a world to gain here. Industry is in need, and academia wants to deliver. However, what is delivered, does not appear to be the right answer. Hence, the answer to the initially posed question, in my opinion, clearly is not positive. The academic community is doing great things, however, the uptake is slow. In other words, we have been doing things right, that is, correct, but the question more is, did we do the right things? It is too easy to say that industry is to blame, because they are not accepting what we deliver and say is good for them.

Why is it that "our methods" are not being used? Why do industrial developers not use what we deliver? Again, there is no definitive answer to this, however, I again would like to make a few observations, that might help in bringing these worlds closer together. As before, the list can be made longer, the current items might not be all orthogonal, still they do provide some insight.

1. The "modeling and analysis" community, that is, we, have become disconnected from the computer systems community. In the 1960's and 1970's, the researchers working on multi-programmed time-sharing computer systems and computer networks, respectively, also did make the models needed to dimension them, see, e.g., [6]. Also the early work on what are now called formal methods, was done by researchers developing protocol systems, in the 1980's, see, e.g., [1,7]. The models and techniques being developed, were clearly developed with a "market pull". By now, it appears we have moved much more towards a "technology push" model, with almost completely disjunct communities and conferences, pushing methods and techniques to the market that are not being bought, for various reasons, be it complexity of use, or simply because they do not have the right functionality. Should we be surprised that our methods are not used?
2. Academic key performance indicators (KPI's), especially those from the sciences, do force researchers to publish many papers, at highly ranked conferences and in top-quality journals. Being part of a community in which one fits well and in which the work is well received, is part of academic survival. It is unfair to solely blame researchers for this. However, there is a side effect that cannot be ignored either.
3. A large part of computer science research has developed itself strongly along the science-axis. The *analysis* of (existing) systems is prevalent. However, the engineering and constructive side is crucial to industry, as this is the line along which value is created. In the end, ICT systems are man-made systems that need to be built, preferably using solid engineering principles. A revitalization of computer engineering, or probably the science of computer engineering, is needed.
4. The study of quantitative properties of computer systems and to develop appropriate modeling and analysis techniques for that, does require an active attitude towards measurements and experimentation, for at least three reasons: (i) to calibrate the system models (parametrization), (ii) to validate

the models themselves, and (iii) to validate modeling approaches. We see too little of this [12]; if we do not validate our models and modeling techniques ourselves, can we expect others to simply believe us and buy them?

5. Going out there, that is, working on real industrial cases is very challenging. Many researchers do feel safer at home, working on the models and techniques they have been working on for a long time, thus avoiding to enter unknown territory. Of course, for an individual Ph.D.-student—the work horse of modern research—it is very risky to embark on a project that is largely industry-driven. At the same time, industry is not always as open as needed to embark on such joint projects. Both professors and senior researchers and developers in industry should take their responsibility in progressing the field.
6. Industrial practice asks for extreme scalability of methods and techniques. The community made great improvements here, especially in the field of symbolic functional verification. We have to go a long way for scalable quantitative analysis, let alone for synthesis while preserving quantitative constraints. Let these needs be leading! This might mean, that paradigm shifts are needed, e.g., moving to mean-field analysis, instead of exact explicit state-space analysis.
7. True industrial use, that is, in system development and not just in research departments, of the techniques and tools we like so much, does require the embedding in daily work routines. For many industries, this means that a nice such-and-such tool, will not be used if it is not part of, e.g., the Matlab Simulink toolset, or cannot be connected to IBM's Rational Rhapsody family of products. Your tool is not their tool! At best, our tools are seen as "engine" in some larger process chain. We cannot deny this. How many academic researchers know which tools are being used in industrial practice? How different is this from other engineering disciplines?

6 Opportunities!

Having made two sets of observations, as well as an analysis, helps us in determining how to improve things. In the end, we all want to have impact with our work, to have the story that our work is being used extensively in the design of new series of products by major industries.

Here, even more than before, I am very careful. I do not have the final recipe, however, I do have some thoughts, which I like to share with you, before I address a way-of-working we execute and advocate at ESI: the industry-as-lab approach.

Please note that my thoughts below do not exclude at all good fundamental research. However, I do say that the balance between fundamental and applied research should probably be shifted, in the engineering discipline we should like to be. Probably the difference between applied and fundamental research is overemphasized here; maybe there is just good and not-so-good research.

1. Wouldn't it be great to see our tools and techniques have true impact? In my opinion this will much more easily happen if we do incorporate industry in

our research projects more firmly, right from the start. This does go beyond doing some realistic industrial case study at the end of a 4-year project. What is needed is industrial involvement in the problem definition and a continuous dialogue between the problem owner (industry) and the solution provider. This does require serious investments from either side.

2. If I would be a medical doctor working in an academic medical centre, I would teach, I would do research, and I would really treat people, even operate them. A similar mix of activities is often seen in Architecture or in Engineering faculties. How rare is this in computer science! It might be an idea to have sabbaticals more often across borders, that is, an academic really working in a development lab in industry, and a computer system of software engineer spending a semester in academia. National funding agencies could accommodate this. In the Netherlands, the Ministry of Economic Affairs supported the latter scheme during the crisis year 2010. This could even taken further, in that such exchanges form part of the human capital agenda of industries and academia alike.
3. The Dutch Technology Foundation STW does foster programs for joint research, which are being defined by committees with members from industry and academia. Moreover, industry does invest in these programs in that they cater for 50% of the costs. In doing so, the problem owner is actively involved. This leads to challenging projects, in which both industry and academia have to deliver.

Finally, at ESI we have successfully followed a scheme which we have called ***industry-as-lab*** after a model proposed by Potts for the software engineering field [11]. At ESI we tailored it to the high-tech embedded systems field, encouraged by new insights in open innovation [3].

Potts observed that most research projects in software engineering follow the traditional research-then-transfer paradigm. In a way, this is a different wording for "technology push". However, the effective result of such projects is often questionable; Potts even classifies some of the typical project goals simply as naive. According to Potts, the industry-as-lab approach sacrifices revolution, but strongly fosters evolution. Industry does not like revolution most of the time, it does like evolution. The key idea is that industry is involved in problem identification at the outset, and jointly with academic partners forms a project description and consortium; this is a true market pull. Interaction between industry and academic partners is intensive, with Ph.D.-students spending time in industry every week, meeting system and software engineers, joining meetings to better understand what the true problems are, and to show what can be learned or gained from more academic approaches. Note that this does go well beyond the industry-academic interaction in many European projects, where project partners meet a couple of times per year, and only interact superficially. Industry-as-lab allows for a continuous large-scale (experimental) validation of research work. It changes the sequential research-*then*-transfer paradigm into a continuous research-***and***-transfer paradigm, allowing very short feedback cycles.

The industry-as-lab way of working is very challenging, but very rewarding as well, leading to good research results achieved under realistic constraints, hence, leading to more direct applicability. Being able to state that one's newly created technique is really used in industry for developing their new so-and-so product, also helps in gaining academic credits; at ESI, we have seen many examples of that.

7 Epilogue

In this short paper I have addressed the question how well the formal modeling and analysis techniques for quantitative system properties, as we know and like time, are taken up in industry. By carefully observing what happens in academia and industry, I have to conclude that despite great developments in academia, the industrial uptake is disappointing. And understandably so, I am afraid. But this is not to say that this cannot improve. Instead, I proposed a number of ways to improve the required interaction in order to shift from a technology push situation, to a market pull situation. The industry-as-lab paradigm, as practiced by ESI, implements such a shift, thereby truly bridging between academia and industry in an open innovation setting.

References

1. Bolognesi, T., Brinksma, E.: Introduction to the ISO Specification Language LOTOS. Computer Networks 14, 25–59 (1987)
2. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Performance Evaluation and Model Checking Join Forces. Communications of the ACM 53(9), 76–85 (2010)
3. Chesbrough, H.W.: The Era of Open Innovation. MIT Sloan Management Review 44(3), 35–41 (2003)
4. The Embedded Systems Institute Strategic Research Agenda 2011 Forthcoming fall (2011), http://www.esi.nl/
5. Thomas, N., Juiz, C. (eds.): EPEW 2008. LNCS, vol. 5261. Springer, Heidelberg (2008)
6. Frenkel, K.A.: Big Blue's Time-Sharing Pioneer. Communications of the ACM 30(10), 824–828 (1987)
7. Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice Hall, Englewood Cliffs (1990)
8. Proceedings IEEE MASCOTS. IEEE CS Press, London (September 2009)
9. Proceedings of the International Conference on Operations Research. Springer, München (September 2010)
10. My DBLP page, http://www.informatik.uni-trier.de/ ley/db/indices/a-tree/h/Haverkort:Boudewijn_R=.html
11. Potts, C.: Software-Engineering Research Revisited. IEEE Software 10(5), 19–28 (1993)
12. Tichy, W.F.: Should Computer Scientists Experiment More? IEEE Computer 31(5), 32–40 (1998)

Interfaces for Control Components

Rajeev Alur

University of Pennsylvania

Modern software engineering heavily relies on clearly specified interfaces for separation of concerns among designers implementing components and programmers using those components. The need for interfaces is evident for assembling complex systems from components, but more so in control applications where the components are designed by control engineers using mathematical modeling tools and used by software executing on digital computers. However, the notion of an interface for a control component must incorporate some information about timing, and standard programming languages do not provide a way of capturing such resource requirements.

This talk will describe how finite automata over infinite words can be used to define interfaces for control components. When the resource is allocated in a time-triggered manner, the allocation from the perspective of an individual component can be described by an infinite word over a suitably chosen alphabet. The control engineer can express the interface of the component as an omega-regular language that contains all schedules that meet performance requirements. The software must ensure, then, that the runtime allocation is in this language. The main benefit of this approach is composability: conjoining specifications of two components corresponds to a simple language-theoretic operation on interfaces. We have demonstrated how to automatically compute automata for performance requirements such as exponential stability and settling time for the LQG control designs. The framework is supported by a toolkit, RTComposer, that is implemented on top of Real Time Java. The benefits of the approach will be demonstrated using applications to wireless sensor/actuator networks based on the WirelessHART protocol and to distributed control systems based on the Control Area Network (CAN) bus.

This talk is based on research reported in [3,2,1,4]. This research is partially supported by NSF awards CNS 0931239 and CCF 0915777.

References

1. Alur, R., D'Innocenzo, A., Johansson, K., Pappas, G., Weiss, G.: Modeling and analysis of multi-hop control networks. In: Proc. 15th IEEE RTAS (2009)
2. Alur, R., Weiss, G.: RTComposer: a framework for real-time components with scheduling interfaces. In: Proc. 8th EMSOFT, pp. 159–168 (2008)
3. Weiss, G., Alur, R.: Automata based interfaces for control and scheduling. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 601–613. Springer, Heidelberg (2007)
4. Weiss, G., Fischmeister, S., Anand, M., Alur, R.: Specification and analysis of network resource requirements of control systems. In: Majumdar, R., Tabuada, P. (eds.) HSCC 2009. LNCS, vol. 5469, pp. 381–395. Springer, Heidelberg (2009)

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, p. 25, 2011.

Time-Bounded Verification of CTMCs against Real-Time Specifications*

Taolue Chen, Marco Diciolla, Marta Kwiatkowska, and Alexandru Mereacre

Department of Computer Science, Oxford University,
Wolfson Building, Parks Road, Oxford, OX1 3QD, United Kingdom

Abstract. In this paper we study time-bounded verification of a finite continuous-time Markov chain (CTMC) $\mathcal{C}$ against a real-time specification, provided either as a metric temporal logic (MTL) property φ, or as a timed automaton (TA) $\mathcal{A}$. The key question is: what is the probability of the set of timed paths of $\mathcal{C}$ that satisfy φ (or are accepted by $\mathcal{A}$) over a time interval of fixed, bounded length? We provide approximation algorithms to solve these problems. We first derive a bound N such that timed paths of $\mathcal{C}$ with at most N discrete jumps are sufficient to approximate the desired probability up to ε. Then, for each discrete (untimed) path σ of length at most N, we generate timed constraints over variables determining the residence time of each state along σ, depending on the real-time specification under consideration. The probability of the set of timed paths, determined by the discrete path and the associated timed constraints, can thus be formulated as a multidimensional integral. Summing up all such probabilities yields the result. For MTL, we consider both the continuous and the pointwise semantics. The approximation algorithms differ mainly in constraints generation for the two types of specifications.

1 Introduction

Verification of *continuous-time Markov chains* (CTMCs) has received much attention in recent years [8]. Thanks to considerable improvements of algorithms, (symbolic) data structures and abstraction techniques, CTMC model checking has emerged as a valuable analysis technique. Aided by powerful software tools, it has been adopted by researchers from, e.g., systems biology, queuing networks and dependability.

The focus of CTMC model checking has primarily been on checking stochastic versions of the *branching-time* temporal logic CTL, such as CSL [7]. The verification of LTL properties reduces to applying well-known algorithms [33,18] to embedded discrete-time Markov chains (DTMCs). Linear-time properties equipped with timing constraints have only recently been considered. In particular, [16,17] treat linear *real-time* specifications that are given as *deterministic timed automata* (DTA). These include properties of the form, “what is the probability to reach a given target state within the deadline, while avoiding unsafe states and not staying too long in any of the dangerous states on the way?”. Such properties cannot be expressed in CSL nor in its dialects [6,19]. Model checking DTA properties can be done by a reduction to computing the

* This work is supported by the ERC Advanced Grant VERIWARE.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 26–42, 2011.

reachability probability in a *piecewise deterministic Markov process*, based on the product construction between the CTMC and DTA [17,11]. It remains a challenge to tackle more general real-time specifications like *Metric Temporal Logics* ([4,24], MTL), or *nondeterministic Timed Automata* (TA, [1]). The main difficulty lies in the fact that one cannot easily define a stochastic process out of the CTMC and the MTL formula (or TA), due to the inherent nondeterminism arising from these specifications. The obstacle is somehow fundamental, as it is known that *deterministic* TA are lacking expressiveness compared to their nondeterministic variants or MTL.

Recently, we have seen increasing emphasis on *timed-bounded verification* [27]. Here, "time-bounded" means restricting the modeling and verification efforts to some bounded interval of time, which itself can be taken as a parameter. In verification, queries are phrased over time intervals of fixed, bounded duration. Note that, differently from bounded model checking, which restricts the total number of allowable events (called discrete jumps in this paper), time-bounded verification restricts the total duration under consideration, but *not* the number of events, which can still be unboundedly large owing to the density of time.[1] Instances of time-bounded verification have been considered in the context of stochastic and/or real-time systems [30,9,23,20] and recently studied systematically [27,22]; see [29] for an introduction, where it is argued that the restriction on total duration is very natural for real-time systems.

Inspired by this recent progress, we study the time-bounded verification problem of a CTMC $\mathcal{C}$, against a real-time specification provided as either an MTL formula φ, or as a TA $\mathcal{A}$. The key question is: what is the probability of the set of timed paths of $\mathcal{C}$ that satisfy φ (or are accepted by $\mathcal{A}$) over a fixed time interval $[0, T]$ where $T \in \mathbb{R}_{>0}$? We provide approximation algorithms to solve these problems. Given any $\varepsilon > 0$ a priori, we first derive a bound N such that it is sufficient only to consider timed paths of $\mathcal{C}$ with at most N discrete jumps to approximate the desired probability up to ε. Then, for each *discrete* (untimed) path σ of $\mathcal{C}$ of length at most N, we generate a family of linear constraints, $\mathcal{S}$, over variables determining the residence time of each state in σ. The discrete path σ, together with the associated timing constraints $\mathcal{S}$, determines a set of *timed* paths of $\mathcal{C}$, each of which satisfies φ (or is accepted by $\mathcal{A}$). The probability of this set of timed paths can be formulated as a multidimensional integral, which can be calculated by Laplace transforms, together with an application of the inclusion-exclusion principle. Summing up all such probabilities yields the desired result. Notice that, in the current paper, we consider both the *continuous* and the *pointwise* semantics of MTL (see, e.g. [14]). The approximation algorithms differ mainly in constraints generation for different types of specifications. The family of *linear* constraints are desirable, since we can apply the efficient algorithm for computing the volumes of convex polyhedra [25]. For MTL under the pointwise semantics and TA specifications, constraint generation is relatively easy, while for MTL under the continuous semantics it is more involved. To this end, we first derive constraints in terms of first-order theory of $(\mathbb{R}, +, -, 0, 1, \leq)$, then the Fourier-Motzkin elimination procedure [31, pp.155-156] is

[1] Readers should note that we later bound the number of discrete jumps as an *approximation technique*. This owes to the definition of CTMCs and is irrelevant to the original definition of time-bounded verification.

applied to obtain desired linear constraints. We believe these results are of independent interest, as they have potential usage in domains such as runtime verification.

The approach we take in this paper is quite different from existing results in the literature. Known results can only deal with simpler real-time properties, or are based on *deterministic* property specifications (e.g. DTA). Our technique is based on path exploration of CTMCs, together with a novel analytic methodology to reduce computing the probabilities to a multi-dimensional integral over convex polyhedra. To the best of our knowledge, this is the first work addressing verification of CTMCs against MTL formulas or non-deterministic timed automata.

Related work. Model checking CTMCs against linear real-time specifications has received scant attention so far. To our knowledge, this issue has only been (partially) addressed in [16,6,19]. Baier *et al.* [6] define the logic asCSL where path properties are characterized by (time-bounded) regular expressions over actions and state formulas. The truth value of path formulas depends not only on the available actions in a given time interval, but also on the validity of certain state formulas in intermediate states. asCSL is strictly more expressive than CSL [6]. Model checking asCSL is performed by representing the regular expressions as finite-state automata, followed by computing time-bounded reachability probabilities in the product of CTMC $\mathcal{C}$ and this automaton. In CSL^{TA} [19], time constraints of until modalities are specified by a single-clock DTA; the resulting logic is at least as expressive as asCSL [19]. The combined behavior of $\mathcal{C}$ and the DTA $\mathcal{A}$ is interpreted as a Markov renewal process, and model checking CSL^{TA} is reduced to computing the reachability probabilities in a DTMC whose transition probabilities are given by subordinate CTMCs.

Due to space restriction, all the proofs are omitted in the current paper. We refer the readers to [15] for the full proofs, more explanation, and examples.

2 Preliminaries

2.1 Continuous-Time Markov Chains

Given a set $\mathcal{H}$, let Pr: $\mathcal{F}(\mathcal{H}) \to [0,1]$ be a *probability measure* on the measurable space $(\mathcal{H}, \mathcal{F}(\mathcal{H}))$, where $\mathcal{F}(\mathcal{H})$ is a σ-algebra over $\mathcal{H}$. Let $\mathit{Distr}(\mathcal{H})$ denote the set of probability measures on this measurable space.

Definition 1 (CTMC). *A* (*labeled*) *continuous-time Markov chain* (CTMC) *is a tuple* $\mathcal{C} = (S, \text{AP}, L, \alpha, \mathbf{P}, E)$ *where* S *is a finite set of states;* AP *is a finite set of atomic propositions;* $L : S \to 2^{\text{AP}}$ *is the labeling function;* $\alpha \in \mathit{Distr}(S)$ *is the initial distribution;* $\mathbf{P} : S \times S \to [0,1]$ *is a stochastic matrix; and* $E : S \to \mathbb{R}_{\geq 0}$ *is the exit rate function.*

In a CTMC $\mathcal{C}$, state residence times are *exponentially* distributed. More precisely, the residence time X of a state $s \in S$ is a random variable governed by a nonnegative exponential distribution with parameter $E(s)$ (written as $X \sim \text{Exp}(E(s))$). Hence, the probability to exit state s in t time units (t.u. for short) is given by $\int_0^t E(s) \cdot e^{-E(s)\tau} d\tau$. Furthermore, the probability to take the transition from s to s' in t t.u. equals $\mathbf{P}(s, s') \cdot \int_0^t E(s) \cdot e^{-E(s)\tau} d\tau$.

Definition 2. *Given a* CTMC $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$*, we define the following notions.*

- *A (finite)* discrete *path* $\sigma = s_0 \to s_1 \to s_2 \to \ldots$ *is a (finite) sequence of states; we define* σ_i *to be the state* s_i*, and* σ^i *to be the prefix of length* i *of* σ*.*
- *A (finite)* timed *path* $\rho = s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} s_2 \xrightarrow{x_2} \ldots$*, where* $x_i \in \mathbb{R}_{>0}$ *for each* $i \geq 0$*, is a sequence starting in state* s_0*; we define* $|\rho|$ *to be the length of a* finite *timed path* ρ*;* $\rho[n] := s_n$ *is the* n*-th state of* ρ *and* $\rho\langle n\rangle := x_n$ *is the time spent in state* s_n*; let* $\rho@t$ *be the state occupied in* ρ *at time* $t \in \mathbb{R}_{\geq 0}$*, i.e.* $\rho@t := \rho[n]$*, where* n *is the smallest index such that* $\sum\limits_{i=0}^{n} \rho\langle i\rangle \geq t$*.*
- *Given a finite discrete path* $\sigma = s_0 \to s_1 \to \cdots \to s_{n-1}$ *of length* n *and* $x_0, \ldots, x_{n-1} \in \mathbb{R}_{>0}$*, define* $\sigma[x_0, \ldots, x_{n-1}]$ *to be the finite timed path* ρ *such that* $\rho[i] := s_i$ *and* $\rho\langle i\rangle := x_i$ *for each* $0 \leq i < n$*.*
- *Let* $\varGamma$ *be the set of* n*-tuples* $(x_0, \ldots, x_{n-1}) \in \mathbb{R}^n_{>0}$*, then* $\sigma[\varGamma] = \{\sigma[x_0, \ldots, x_{n-1}] \mid (x_0, \ldots, x_{n-1}) \in \varGamma\}$*.*
- *Given a finite (resp. infinite) discrete path* σ *and a finite (resp. infinite) timed path* ρ*, we say* σ *is the* skeleton *of* ρ *if for each* $i \geq 0$*,* $\sigma_i = \rho[i]$*. We write* $\mathbb{S}(\rho)$ *for the skeleton of* ρ*, and for a set of (finite or infinite) timed paths* $\varXi$*, we write* $\mathbb{S}(\varXi) = \{\mathbb{S}(\rho) \mid \rho \in \varXi\}$*.*
- *Given a* finite *discrete path* σ*, we define* $C_d(\sigma) = \{\sigma\sigma' \mid \sigma' \textit{ is an infinite discrete path}\}$ *to be the set of all infinite discrete paths with the same common prefix* σ*.*

Intuitively, a timed path ρ suggests that the CTMC $\mathcal{C}$ starts in state s_0 and stays in this state for x_0 t.u., and then jumps to state s_1, staying there for x_1 t.u., and then jumps to s_2 and so on. An example timed path is $\rho = s_0 \xrightarrow{3} s_1 \xrightarrow{2} s_0 \xrightarrow{1.5} s_1 \xrightarrow{3.4} s_2 \ldots$ with $\rho[2] = s_0$ and $\rho@4 = \rho[1] = s_1$.

Let $\mathit{Paths}^{\mathcal{C}}$ denote the set of infinite timed paths in the CTMC $\mathcal{C}$, and $\mathit{Paths}^{\mathcal{C}}(s)$ the set of infinite timed paths in $\mathcal{C}$ that start in s. Given a time bound $T \in \mathbb{R}_{\geq 0}$ and $N \in \mathbb{N} \cup \{\infty\}$, we define $\mathit{Paths}^{\mathcal{C}}_{T,<N}(s) = \big\{\rho \in \mathit{Paths}^{\mathcal{C}}(s) \mid \exists k.0 \leq k \leq N-1$ and $\sum_{i=0}^{k} \rho\langle i\rangle \geq T\big\}$, to be the set of all timed paths with *at most* $N-1$ discrete jumps in time interval $[0, T]$; and $\mathit{Paths}^{\mathcal{C}}_{T,\geq N}(s) = \big\{\rho \in \mathit{Paths}^{\mathcal{C}}(s) \mid \exists k.0 \leq k \leq N-1,$ and $\sum_{i=0}^{k} \rho\langle i\rangle \leq T\big\}$, to be the set of all timed paths with *at least* N jumps in $[0, T]$.

For notational simplicity we will omit the superscript $\mathcal{C}$ when appropriate and also we write $\mathit{Paths}^{\mathcal{C}}_{T}$ instead of $\mathit{Paths}^{\mathcal{C}}_{T,\leq\infty}$ for the set of all timed paths with an arbitrary number of jumps in $[0, T]$. The definition of a Borel space on timed paths through CTMCs follows [7]. A CTMC $\mathcal{C}$ yields a probability measure $\Pr^{\mathcal{C}}$ on $\mathit{Paths}^{\mathcal{C}}$ as follows. Let $s_0, \ldots, s_k \in S$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for $0 \leq i < k$ and $I_0, \ldots, I_{k-1}$ be nonempty intervals in $\mathbb{R}_{\geq 0}$. Let $C(s_0, I_0, \ldots, I_{k-1}, s_k)$ denote the *cylinder set* consisting of all $\rho \in \mathit{Paths}(s_0)$ such that $\rho[i] = s_i$ $(i \leq k)$, and $\rho\langle i\rangle \in I_i$ $(i < k)$. $\mathcal{F}(\mathit{Paths}(s_0))$ is the smallest σ-algebra on $\mathit{Paths}(s_0)$ which contains all sets $C(s_0, I_0, \ldots, I_{k-1}, s_k)$ for all state sequences $(s_0, \ldots, s_k) \in S^{k+1}$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for $(0 \leq i < k)$ where $I_0, \ldots, I_{k-1}$ range over all sequences of nonempty intervals in $\mathbb{R}_{\geq 0}$.

The probability measure $\Pr^{\mathcal{C}}$ on $\mathcal{F}(Paths(s_0))$ is the unique measure defined by induction on k by $\Pr^{\mathcal{C}}(C(s_0)) = \alpha(s_0)$ and for $k > 0$:

$$\Pr^{\mathcal{C}}(C(s_0, I_0, \ldots, I_{k-1}, s_k)) = \Pr^{\mathcal{C}}(C(s_0, I_0, \ldots, I_{k-2}, s_{k-1})) \\ \times \int_{I_{k-1}} \mathbf{P}(s_{k-1}, s_k) E(s_{k-1}) \cdot e^{-E(s_{k-1})\tau} d\tau.$$

In general, computing the probability of a cylinder set with k intervals $I_0 \ldots I_{k-1}$ (i.e. k discrete jumps) reduces to calculating k integrals over $I_0 \ldots I_{k-1}$.

2.2 Metric Temporal Logic

Definition 3 (Syntax of MTL). *Let* AP *be an arbitrary nonempty, finite set of atomic propositions. Let* $I = [a, b]$ *be an interval such that* $a, b \in \mathbb{N} \cup \{\infty\}$. *The* Metric Temporal Logic *is inductively defined as:* $\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2$ *, where* $p \in$ AP *and* φ_1, φ_2 *are* MTL *formulas.*

We introduce two time-bounded semantics for MTL, as follows.

Definition 4 (Continuous Semantics). *Given an* MTL *formula* φ*, a time bound* T*, a timed path* ρ *and a variable* $t \in \mathbb{R}_{\geq 0}$*, the satisfaction relation* $(\rho, t) \models^c_T \varphi$ *is inductively defined as follows:*

$$\begin{array}{ll}
(\rho, t) \models^c_T p & \Leftrightarrow p \in L(\rho@t) \wedge t \leq T \\
(\rho, t) \models^c_T \neg\varphi_1 & \Leftrightarrow (\rho, t) \not\models^c_T \varphi_1 \\
(\rho, t) \models^c_T \varphi_1 \wedge \varphi_2 & \Leftrightarrow (\rho, t) \models^c_T \varphi_1 \wedge (\rho, t) \models^c_T \varphi_2 \\
(\rho, t) \models^c_T \varphi_1 \mathcal{U}_I \varphi_2 & \Leftrightarrow \exists t'.\ t \leq t' \leq T \textit{ s.t. } t' - t \in I \wedge (\rho, t') \models^c_T \varphi_2 \wedge \\
& \quad \forall t''.\ t \leq t'' < t' \Rightarrow (\rho, t'') \models^c_T \varphi_1
\end{array}$$

where $p \in$ AP *and* φ_1, φ_2 *are* MTL *formulas.*

Definition 5 (Pointwise Semantics). *Given an* MTL *formula* φ*, a time bound* T*, a timed path* ρ *and* $i \in \mathbb{N}$*, the satisfaction relation* $(\rho, i) \models^p_T \varphi$ *is inductively defined as follows:*

$$\begin{array}{ll}
(\rho, i) \models^p_T p & \Leftrightarrow p \in L(\rho[i]) \wedge \sum_{k=0}^{i} \rho\langle k\rangle \leq T \\
(\rho, i) \models^p_T \neg\varphi_1 & \Leftrightarrow (\rho, i) \not\models^p_T \varphi_1 \\
(\rho, i) \models^p_T \varphi_1 \wedge \varphi_2 & \Leftrightarrow (\rho, i) \models^p_T \varphi_1 \wedge (\rho, i) \models^p_T \varphi_2 \\
(\rho, i) \models^p_T \varphi_1 \mathcal{U}_I \varphi_2 & \Leftrightarrow \exists i'.\ i \leq i' \textit{ s.t. } \sum_{k=i}^{i'} \rho\langle k\rangle \in I \wedge (\rho, i') \models^p_T \varphi_2 \wedge \\
& \quad \forall i''.\ i \leq i'' < i' \Rightarrow (\rho, i'') \models^p_T \varphi_1
\end{array}$$

where $p \in$ AP*,* φ_1*,* φ_2 *are* MTL *formulas and* $i', i'' \in \mathbb{N}$.

2.3 Timed Automata

Let $\mathcal{X} = \{x_1, \ldots, x_p\}$ be a set of nonnegative real-valued variables called *clocks*. An $\mathcal{X}$-valuation is a function $\eta : \mathcal{X} \to \mathbb{R}_{\geq 0}$ assigning to each variable $x \in \mathcal{X}$ a nonnegative real value $\eta(x)$. Let $\mathcal{V}(\mathcal{X})$ denote the set of all valuations over $\mathcal{X}$. A *clock constraint* on $\mathcal{X}$, denoted by g, is a conjunction of expressions of the form $x \bowtie c$ for $x \in \mathcal{X}$, $\bowtie \in \{<, \leq, >, \geq\}$ and $c \in \mathbb{N}$. Let $\mathcal{B}(\mathcal{X})$ denote the set of clock constraints over $\mathcal{X}$. An $\mathcal{X}$-valuation η *satisfies* constraint $x \bowtie c$, denoted $\eta \models x \bowtie c$, if and only if $\eta(x) \bowtie c$; it satisfies a conjunction of such expressions if and only if η satisfies all of them. Let $\mathbf{0}$ denote the valuation that assigns 0 to all clocks. For a subset $X \subseteq \mathcal{X}$, the reset of X, denoted $\eta[X := 0]$, is the valuation η' such that $\forall x \in X.\ \eta'(x) := 0$ and $\forall x \notin X.\ \eta'(x) := \eta(x)$. For $\delta \in \mathbb{R}_{\geq 0}$ and $\mathcal{X}$-valuation η, $\eta + \delta$ is the $\mathcal{X}$-valuation η'' such that $\forall x \in \mathcal{X}.\ \eta''(x) := \eta(x) + \delta$, which implies that all clocks proceed at the same speed.

Definition 6 (TA). *A* timed automaton *is a tuple $\mathcal{A} = (\Sigma, \mathcal{X}, Q, q_0, Q_{\boldsymbol{F}}, \rightarrow)$ where Σ is a finite* alphabet*; $\mathcal{X}$ is a finite set of* clocks*; Q is a non empty finite set of* locations *with initial location $q_0 \in Q$; $Q_{\boldsymbol{F}}$ is a set of* final *locations; the relation $\rightarrow \subseteq Q \times \Sigma \times \mathcal{B}(\mathcal{X}) \times \mathbf{2}^{\mathcal{X}} \times Q$ is an* edge relation.

We refer to $q \xrightarrow{a,g,X} q'$ as an *edge*, where $a \in \Sigma$ is an input symbol, the *guard* g is a clock constraint on the clocks of $\mathcal{A}$, X is the set of clocks that must be reset and q' is the successor location. Intuitively, the edge $q \xrightarrow{a,g,X} q'$ asserts that the TA $\mathcal{A}$ can move from location q to location q' when the input symbol is a and the guard g holds, while the clocks in X should be reset when entering q'. In case no guard is satisfied in a location for a given clock valuation, time can progress. For the sake of simplicity we omit invariants from the definition of TAs. However, the results presented here can be easily extended to TAs enhanced with invariants.

Definition 7. *Given a timed automaton $\mathcal{A}$, we define the following notions.*

- *A* discrete *path of $\mathcal{A}$ is a sequence of states $w = q_0 \rightarrow q_1 \ldots \rightarrow q_n \cdots$ where each $q_i \in Q$.*
- *A* timed *path of $\mathcal{A}$ is of the form $\theta = q_0 \xrightarrow{a_0,t_0} q_1 \xrightarrow{a_1,t_1} \ldots q_{n-1} \xrightarrow{a_{n-1},t_{n-1}} q_n \cdots$ such that $\eta_0 = \mathbf{0}$, and for all $i \geq 0$, $a_i \in \Sigma$ and it holds $t_i > 0$, $\eta_i + t_i \models g_i$ where g_i is the guard on the i-th transition, $\eta_{i+1} = (\eta_i + t_i)[X_i := 0]$, where η_i is the clock evaluation when entering q_i. We say that θ* is accepting *if there exists some $n \geq 0$ s.t. $q_n \in Q_{\mathbf{F}}$.*

Definition 8 (Time-bounded Acceptance). *Assume a* CTMC $\mathcal{C} = (S, \mathrm{AP}, L, s_0, \mathbf{P}, E)$ *and a* TA $\mathcal{A} = (2^{AP}, \mathcal{X}, Q, q_0, Q_{\mathbf{F}}, \rightarrow)$. *A* CTMC *timed path $\rho = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots$, is accepted by $\mathcal{A}$ if there* exists *$n \in \mathbb{N}_{>0}$ and a corresponding* TA *finite path: $\theta = q_0 \xrightarrow{L(s_0),t_0} q_1 \xrightarrow{L(s_1),t_1} \ldots q_{n-1} \xrightarrow{L(s_{n-1}),t_{n-1}} q_n$, such that $q_n \in Q_{\boldsymbol{F}}$ and $\sum_{i=0}^{n-1} t_i \leq T$. We write $\rho \models_T \mathcal{A}$ to denote that the* CTMC *timed path ρ is accepted by $\mathcal{A}$.*

Remark 1. It is possible that a single CTMC timed path corresponds to multiple TA accepting paths due to the nondeterminism of TA.

3 A Bound on the Number of Discrete Jumps

In this section, we give a bound on discrete jumps of paths of CTMCs such that, when verifying an MTL formula or TA, one only needs to consider those paths whose discrete jumps number at most N. The intuition is that, for a given time interval $[0,T]$, the probability of the set of timed paths which "jump" very frequently is actually very small. Throughout this section we assume a CTMC $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$.

For any $n \in \mathbb{N}$, we define $V^n(s,x) : S \times \mathbb{R}_{\geq 0} \to [0,1]$ as follows: $V^0(s,x)$=1 and

$$V^{n+1}(s,x) = \int_0^x E(s)e^{-E(s)\tau} \cdot \sum_{s' \in S} \mathbf{P}(s,s') \cdot V^n(s', x-\tau) d\tau \ .$$

Lemma 1. *For all* $N \in \mathbb{N}$, $\Pr^{\mathcal{C}}(\mathit{Paths}^{\mathcal{C}}_{T,\geq N}(s)) = V^N(s,T)$.

We then show how to bound $V^N(s,T)$ analytically. Given a CTMC $\mathcal{C}$, let $\Lambda = \max_{s\in S} E(s)$ and $\epsilon(T,N) = e^{-\Lambda T} \cdot \left(\sum_{i=N}^{\infty} \frac{(\Lambda T)^i}{i!}\right)$.

Lemma 2. $\epsilon(T, N+1) = \int_0^T \Lambda e^{-\Lambda\tau} \cdot \epsilon(T-\tau, N) d\tau$.

Combining Lem. 1 and Lem. 2, we obtain the following.

Theorem 1. *Given a* CTMC $\mathcal{C}$, *a time bound* T *and* $N \in \mathbb{N}$, $\Pr^{\mathcal{C}}(\mathit{Paths}^{\mathcal{C}}_{T,\geq N}) \leq \epsilon(T,N)$.

Proposition 1. *Let* $\varepsilon \in \mathbb{R}_{>0}$ *and* $T \in \mathbb{R}_{\geq 0}$. *For any* $N \geq \Lambda T e^2 + \ln(\frac{1}{\varepsilon})$ *we have that* $\epsilon(T,N) < \varepsilon$.

For instance, given a CTMC $\mathcal{C}$ with 10 states, greatest rate $\Lambda = 100$, error bound $\epsilon = 10^{-2}$ and $T = 1000$, we get that $N \geq 738911$. The maximum number of paths to consider would be 10^N.

Remark 2. Readers who are familiar with Poisson distributions will immediately notice that the bound we obtained is actually the probability that there are at least N Poisson arrivals in an interval of time $[0,T]$, with rate Λ. If the CTMC $\mathcal{C}$ is uniform (i.e., each state of $\mathcal{C}$ has the same exit rate), then one could obtain the bound in a straightforward way. However, for the general case, this cannot be achieved directly. Moreover, we point out here that, in order to verify an MTL formula φ or a TA $\mathcal{A}$, one *cannot* apply the unformization technique, which is used only for transient probability computation.

4 MTL Specifications

In this section we study the problem of model checking CTMCs against MTL properties. Let $\Pr^{\mathcal{C}}_T(\varphi) := \Pr^{\mathcal{C}}(\{\rho \in \mathit{Paths}^{\mathcal{C}}_T \mid (\rho, 0) \models^{\mathrm{c}}_T \varphi\})$ denote the probability that the CTMC $\mathcal{C}$ satisfies the MTL formula φ, for a given time bound T. Notice that, here the definition of $\Pr^{\mathcal{C}}_T(\varphi)$ is for the continuous semantics of MTL. However, we present algorithms to deal with both continuous and pointwise semantics. Instead of computing $\Pr^{\mathcal{C}}_T(\varphi)$, we give a procedure to compute $\Pr^{\mathcal{C}}_{T,<N}(\varphi) := \Pr^{\mathcal{C}}(\mathit{Paths}^{\mathcal{C}}_{T,<N}(\varphi))$ for sufficiently large N which ensures that $\Pr^{\mathcal{C}}_T(\varphi) - \Pr^{\mathcal{C}}_{T,<N}(\varphi) < \varepsilon$ for arbitrarily small $\varepsilon \in \mathbb{R}_{>0}$. This yields an approximation algorithm. The measurability of the set of $\mathit{Paths}^{\mathcal{C}}_{T,<N}(\varphi) := \{\rho \in \mathit{Paths}^{\mathcal{C}}_{T,<N} \mid (\rho,0) \models^{\mathrm{c}}_T \varphi\}$ can be shown as in [32]. Below we present an algorithm to compute $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$. We first give a sketch, and provide the crucial sub-procedures in Sec. 4.1 and Sec. 4.2.

Choose N to get the desired error bound ε. The first step of the algorithm is to choose the smallest N from Prop. 1 such that we get the desired error bound ε.

Compute the product $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$. The basic idea of this step is to exclude those CTMC timed paths which definitely fail φ in order to reduce the number of paths to be analyzed. To this end, we define an LTL formula $\widetilde{\varphi}$ such that, if a discrete path of $\mathcal{C}$ fails $\widetilde{\varphi}$, then any timed path with the discrete path as the skeleton (see Def. 2) must fail φ. This is formally stated in Lem. 3. Notice that since we consider the time-bounded semantics of MTL, we need a variant of acceptance for an infinite discrete word and an LTL formula $\widetilde{\varphi}$, which is given in Def. 9. We then construct an NFA out of $\widetilde{\varphi}$ such that only those finite discrete CTMC paths which are accepted by the NFA are the prefixes of the potential skeletons of timed paths satisfying φ. Then we apply the standard product construction, which suffices to identify those CTMC finite discrete paths analyzed in the next step.

Any MTL formula φ can be transformed into a *positive normal form* containing only two temporal operators: $\mathcal{U}_{[a,b]}$ and $\Box_{[a,b]}$, where $(\rho, t) \models^{\mathrm{c}}_{T} \Box_{[a,b]}\varphi$ iff $\forall t' \in [a,b] \Rightarrow (\rho, t+t') \models^{\mathrm{c}}_{T} \varphi$.

Definition 9 (Bounded Semantics of LTL). *Given an* LTL *formula φ, a* finite *discrete path σ and $i \in \mathbb{N}$, the satisfaction relation $(\sigma, i) \models \varphi$ is inductively defined as follows:*

$$\begin{array}{ll}
(\sigma,i) \models p & \Leftrightarrow p \in L(\sigma_i) \textit{ and } i \le |\sigma| \\
(\sigma,i) \models \neg\varphi_1 & \Leftrightarrow (\sigma,i) \not\models \varphi_1 \\
(\sigma,i) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow (\sigma,i) \models \varphi_1 \wedge (\sigma,i) \models \varphi_2 \\
(\sigma,i) \models \varphi_1 \mathcal{U} \varphi_2 & \Leftrightarrow \exists i'.\ i \le i' \le |\sigma| \textit{ s.t. } (\sigma,i') \models \varphi_2 \wedge \\
 & \quad \forall i''.\ i \le i'' < i' \Rightarrow (\sigma,i'') \models \varphi_1
\end{array}$$

where $p \in$ AP, φ_1, φ_2 are LTL *formulas and $i', i'' \in \mathbb{N}$. For an* infinite *discrete path σ, we define $\sigma \models \varphi$ if there exists some $k \ge 0$ such that the finite discrete path $(\sigma^k, 0) \models \varphi$.*

Given any MTL φ *in positive normal form*, we define an (untimed) LTL formula $\widetilde{\varphi}$ as follows:

$$\begin{array}{ll}
\varphi = p & \Rightarrow \widetilde{\varphi} = p \\
\varphi = \neg p & \Rightarrow \widetilde{\varphi} = \neg p \\
\varphi = \varphi_1 \vee \varphi_2 & \Rightarrow \widetilde{\varphi} = \widetilde{\varphi}_1 \vee \widetilde{\varphi}_2 \\
\varphi = \varphi_1 \wedge \varphi_2 & \Rightarrow \widetilde{\varphi} = \widetilde{\varphi}_1 \wedge \widetilde{\varphi}_2 \\
\varphi = \varphi_1 \mathcal{U}_I \varphi_2 & \Rightarrow \widetilde{\varphi} = \widetilde{\varphi}_1 \mathcal{U} \widetilde{\varphi}_2 \\
\varphi = \Box_I \varphi_1 & \Rightarrow \widetilde{\varphi} = \textsc{true}\, \mathcal{U} \widetilde{\varphi}_1
\end{array}$$

where φ_1 and φ_2 are MTL formulas and $\widetilde{\varphi}_1$ and $\widetilde{\varphi}_2$ are LTL formulas.

Remark 3. In the transformation from the MTL formula φ to LTL formula $\widetilde{\varphi}$ we only define the $\neg$ operator for atomic propositions because φ is already in positive normal form. Notice that we transform $\Box_{[a,b]}\varphi$ into $\textsc{true}\, \mathcal{U}\widetilde{\varphi}$ instead of a seemingly more natural $\Box\varphi$, because otherwise in the next step we would not consider timed paths ρ such that $(\rho, 0) \models \varphi$ while $\mathbb{S}(\rho) \not\models \widetilde{\varphi}$. Such paths do exist. For instance, consider the MTL formula $\Box_{[0,2]}p$ and the path $\rho = s_0 \xrightarrow{2.5} s_1 \cdots$ with $L(s_0) = \{p\}$ and $L(s_1) = \{\neg p\}$. Then $(\rho, 0) \models^{\mathcal{C}}_{T} \Box_{[0,2]}p$ and $\mathbb{S}(\rho) \not\models \Box p$ (but $\mathbb{S}(\rho) \models$ TRUE $\mathcal{U}p$ as we defined). To conclude, one cannot transform $\Box_{[a,b]}$ by simply removing the time constraints $[a, b]$.

Lemma 3. *Let φ be an* MTL *formula and ρ be a timed path in $\mathcal{C}$. We have that*

$$(\rho, 0) \models^{\mathcal{C}}_{T} \varphi \Rightarrow (\mathbb{S}(\rho), 0) \models \widetilde{\varphi}.$$

As the next step, we construct an NFA $\mathcal{A}_{\widetilde{\varphi}}$ which accepts all the prefixes of infinite paths satisfying the formula $\widetilde{\varphi}$ according to Def. 9. The NFA can be obtained by a minor adaptation of the well-known Vardi-Wolper construction [34]. (See [15] for details.) We then build the product of $\mathcal{C}$ and $\mathcal{A}_{\widetilde{\varphi}}$.

Definition 10 (Product $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$). *Given a* CTMC $\mathcal{C} = (S, \mathrm{AP}, L, s_0, \mathbf{P}, E)$ *and an* NFA $\mathcal{A}_{\widetilde{\varphi}} = (Q, 2^{\mathrm{AP}}, \delta, q_0, F)$ *we define the product $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ to be the tuple $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}} = (Loc, l_0, Loc_F, \rightsquigarrow)$ where: $Loc = S \times Q$; $l_0 = \langle s_0, q_0 \rangle$; $Loc_F = S \times F$; $\rightsquigarrow \subseteq Loc \times Loc$ such that*

$$\frac{\mathbf{P}(s, s') > 0 \wedge q \xrightarrow{L(s)} q'}{\langle s, q \rangle \rightsquigarrow \langle s', q' \rangle}.$$

The set of accepted timed paths in $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ is defined by $\Diamond Loc_F$. Notice that we are only interested in the discrete paths of $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$. Therefore, we do not assign probabilities to the transition relation $\rightsquigarrow$ when computing the product. The product is used to check which discrete paths in the CTMC verify the formula $\widetilde{\varphi}$.

Proposition 2. *For any* CTMC $\mathcal{C}$ *and* NFA $\mathcal{A}_{\widetilde{\varphi}}$, $\mathbb{S}(Paths^{\mathcal{C}}_{T}(\varphi)) \subseteq \{C_d(\sigma) \mid \sigma \in \Diamond Loc_F \downharpoonright_1\}$, *where $Loc_F \downharpoonright_1$ is the first component of Loc_F.*

Compute all the discrete paths of $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ of length at most N and calculate the probabilities.

1. Search the graph $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ to get all the discrete accepting paths σ of $\mathcal{C}$ of length at most N;
2. Run Alg. 1 on each discrete path σ of length $n \leq N$ to obtain the system of linear inequalities $\mathcal{S}$;
3. Compute the probability of $\sigma[\mathcal{S}]$ (cf. Sec. 4.2);
4. Sum up all the probabilities for each discrete path to obtain $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$.

4.1 Constraints Generation

We describe the Alg. 1 that takes as input a discrete path σ of length n and an MTL formula φ.[2] The algorithm returns a family of linear constraints $\mathcal{S} = \bigvee_{i \in I} \bigwedge_{j \in J_i} c_{ij}$ where c_{ij} is a linear inequality over the set of variables $t_0, \ldots, t_{n-1}$. Given a system of linear constraints $\mathcal{S}$ we define the set of feasible solutions to be the tuples $(x_0, \ldots, x_{n-1}) \in \mathbb{R}^n$ such that $(x_0, \ldots, x_{n-1}) \in \mathcal{S}$.

[2] The algorithm Alg. 1 evaluates the formula φ for the continuous semantics.

Algorithm 1. Constraints generation for continuous semantics

```
Require: A finite discrete path $\sigma$ of length $n > 0$, an MTL formula $\varphi$ and a time bound $T$
Ensure: Family of linear inequalities $\mathcal{S}$ over $t_0, \dots, t_{n-1}$
  $\mathcal{S}'$ :=Constr_Gen($\sigma$,0,$\varphi$)
  $\mathcal{S}$ :=Fourier_Motzkin($\mathcal{S}'$,$t_0$,...,$t_{n-1}$)
  return $\mathcal{S}$

  Function Constr_Gen($\sigma$,$t$,$\varphi$)
  case($\varphi$) :
    $\varphi = p$                              : return $\left(\bigvee_{k=0}^{n} p \in L(\sigma_k) \wedge \sum_{i=0}^{k} t_i \geq t \wedge \sum_{i=0}^{k-1} t_i < t\right) \wedge t < T$
    $\varphi = \neg\varphi_1$                  : $\mathcal{S}' := \neg$Constr_Gen($\sigma$,$t$,$\varphi_1$)
    $\varphi = \varphi_1 \wedge \varphi_2$     : $\mathcal{S}'$ :=Constr_Gen($\sigma$,$t$,$\varphi_1$) $\wedge$ Constr_Gen($\sigma$,$t$,$\varphi_2$)
    $\varphi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$ : $\mathcal{S}' := \exists t'.\big(t \leq t' < T \wedge t'-t \geq a \wedge t'-t < b \wedge$ Constr_Gen($\sigma$,$t'$,$\varphi_2$)
                                                     $\wedge\ \forall t''.\ t \leq t'' < t' \Rightarrow$ Constr_Gen($\sigma$,$t''$,$\varphi_1$)$\big)$
  return $\mathcal{S}'$
```

The negation of the family of linear constraints is defined in the standard way. First, the algorithm executes the function `Constr_Gen(`σ`,`0`,`φ`)`. The result is a set of constraints $\mathcal{S}'$ in first-order theory of $(\mathbb{R}, +, -, 0, 1, \leq)$. Second, the algorithm executes the Fourier-Motzkin procedure in order to eliminate all existential and universal quantifiers. This results in a family of linear constraints containing only the variables $t_0, \dots, t_{n-1}$.

Theorem 2. *Given a discrete path σ of length n, an* MTL *formula φ and a time bound T, we have that $(\sigma[x_0, \dots, x_{n-1}], 0) \models_T \varphi$ iff $(x_0, \dots, x_{n-1}) \in \mathcal{S}$, where $\mathcal{S}$ is returned by Alg. 1.*

Example 1. Let $\mathcal{C}$ be a CTMC and let σ be the following finite discrete path on $\mathcal{C}$: $\sigma = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3$. Let $a, b \in$ AP, let $L(s_0) = \{a\}, L(s_1) = \{a\}, L(s_2) = \{a, b\}, L(s_3) = \{\varnothing\}$ and let $\varphi = a\,\mathcal{U}_{[1,2]} b$. The first step of Alg. 1 consists of computing `Constr_Gen(`σ`,`0`,`φ`)` which returns the following family of linear constraints $\mathcal{S}'$ (the parenthesis "{" denotes the $\wedge$ between the formulas):

$$\exists t'.\ 0 \leq t' < T \wedge t' \geq 1 \wedge t' < 2 \wedge \begin{cases} t_0 + t_1 + t_2 \geq t' \\ t_0 + t_1 \qquad\ < t' \end{cases} \wedge \tag{1}$$

$$\forall t''.\ 0 \leq t'' < t' \Rightarrow \left(t_0 \geq t'' \vee \begin{cases} t_0 + t_1 \geq t'' \\ t_0 \qquad < t'' \end{cases} \vee \begin{cases} t_0 + t_1 + t_2 \geq t'' \\ t_0 + t_1 \qquad\ < t'' \end{cases} \right). \tag{2}$$

The constraints in Eq. (2) can always be verified given the constraints in Eq. (1). Moreover, after the `Fourier_Motzkin` elimination for t', t'' in $\mathcal{S}'$ we obtain the family of constraints $\mathcal{S}$:

$$\mathcal{S} = \begin{cases} t_0 + t_1 \qquad\ < 2 \\ t_0 + t_1 + t_2 \geq 1 \end{cases}.$$

The system $\mathcal{S}$ can be represented using the matrix notation: $\mathcal{S} := \{\mathbf{t} \in \mathbb{R}^n_{>0} \mid \mathbf{A} \cdot \mathbf{t} \trianglelefteq \mathbf{b}\}$, for a given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, vector $\mathbf{b} \in \mathbb{R}^m$ and $\trianglelefteq \in \{<, \leq\}$. The notation $\mathbb{R}_{>0}$ stands for the semi-closed interval $(0, \infty) \subset \mathbb{R}$. The matrices $\mathbf{A}$, $\mathbf{t}$ and $\mathbf{b}$ in $\mathcal{S}$ are: $\mathbf{A} \in \mathbb{R}^{2\times 3}$, $\mathbf{t} \in \mathbb{R}^3_{>0}$ and $\mathbf{b} \in \mathbb{R}^2$. More specifically:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ -1 & -1 & -1 \end{bmatrix}; \mathbf{t} = \begin{bmatrix} t_0 \\ t_1 \\ t_2 \end{bmatrix}; \mathbf{b} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}.$$

In Alg. 2 we present a procedure which generates a family of linear constraints from a given MTL formula φ under the pointwise semantics. Notice that we do not need to use the `Fourier_Motzkin` elimination procedure, as the family of constraints obtained from `Constr_Gen(`σ`,0,`φ`)` contains no quantifiers.

Algorithm 2. Constraints generation for pointwise semantics

```
Require: A finite discrete path $\sigma$ of length $n > 0$, an MTL formula $\varphi$ and a time bound $T$
Ensure: Family of linear inequalities $\mathcal{S}$ over $t_0, \ldots, t_{n-1}$
  return Constr_Gen($\sigma$,0,$\varphi$)

  Function Constr_Gen($\sigma$,$i$,$\varphi$)
  case($\varphi$) :
    $\varphi = p$                          : if $p \in L(\sigma_i)$ return $\sum_{k=0}^{i} t_k \leq T$ else return false
    $\varphi = \neg\varphi_1$              : $\mathcal{S} := \neg$Constr_Gen($\sigma$,$i$,$\varphi_1$)
    $\varphi = \varphi_1 \wedge \varphi_2$ : $\mathcal{S}$ :=Constr_Gen($\sigma$,$i$,$\varphi_1$) $\wedge$ Constr_Gen($\sigma$,$i$,$\varphi_2$)
    $\varphi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$ : $\mathcal{S} := \big(\bigvee_{i'=i}^{n}$ Constr_Gen($\sigma$,$i'$,$\varphi_2$) $\wedge\ a \leq \sum_{k=i}^{i'} t_k \leq b\ \wedge$
                                             $(\bigwedge_{i''=i}^{i'-1}$Constr_Gen($\sigma$,$i''$,$\varphi_1$)$)\big)$
  return $\mathcal{S}$
```

Let $\mathcal{S}$ be the family of linear constraints obtained from Alg. 1 and 2. $\mathcal{S}$ is always defined as a *union* of convex polyhedra in $\mathbb{R}^n$, i.e., $\mathcal{S} = \bigvee_{i \in I} \bigwedge_{j \in J_i} c_{ij}$ where, for each $i \in I$, $\bigwedge_{j \in J_i} c_{ij}$ is a convex polyhedron.

4.2 Computing Probabilities

Given a CTMC $\mathcal{C}$, a discrete path σ of length N and the family of linear constraints $\mathcal{S}(t_0, \ldots, t_{N-1})$ obtained from Alg. 1, the main task of this section is to compute the probability of $\sigma[\mathcal{S}]$, i.e., $\Pr^{\mathcal{C}}(\sigma[\mathcal{S}])$. To this end, we first add more constraints to $\mathcal{S}$, namely, for $\mathcal{S} = \bigvee_{i \in I} \bigwedge_{j \in J_i} c_{ij}$ we obtain

$$\overline{\mathcal{S}} = \bigvee_{i \in I} \left(\bigwedge_{j \in J_i} c_{ij} \wedge (t_0 + \ldots + t_{N-1} > T \wedge t_0 + \ldots + t_{N-2} < T) \wedge \bigwedge_{0 \leq k < N} t_k > 0 \right).$$

These new constraints are used to ensure that there are *exactly* N discrete jumps during the time interval $[0, T]$, and that each residence time is positive.

Now we have N random variables $t_0, \cdots, t_{N-1}$, corresponding to the residence time of each state σ_i for $i \leq N$. The probability $\Pr^{\mathcal{C}}(\sigma[\overline{\mathcal{S}}])$ is thus formulated as the joint probability $\Pr^{\mathcal{C}}(\overline{\mathcal{S}}(t_0, \cdots, t_{N-1}))$, where $t_i \sim \text{Exp}(E(\sigma_i))$ for each $0 \leq i < N$, and $t_0, \cdots, t_{N-1}$ are bounded by the family of linear constraints $\overline{\mathcal{S}}$. The value of the joint probability can be computed through the following multidimensional integration:

$$\Pr^{\mathcal{C}}(\sigma[\overline{\mathcal{S}}]) = \underbrace{\int \cdots \int}_{N}{}_{\overline{\mathcal{S}}(\tau_0,\ldots,\tau_{N-1})} \prod_{i=0}^{N-1} E(s_i) \cdot P(s_i, s_{i+1}) \times e^{-E(s_i)\tau_i} d\tau_i. \tag{3}$$

Proposition 3 ([21]). *Consider any family of linear inequalities $\overline{\mathcal{S}} = \bigvee_{i \in I} \bigwedge_{j \in J_i} c_{ij}$. For each $i \in I$, we can write $\bigwedge_{j \in J_i} c_{ij}$ in matrix form $\mathbf{A_i} \cdot \mathbf{t} \trianglelefteq \mathbf{b_i}$ where $\trianglelefteq \in \{<, \leq\}$, and $\bigwedge_{j \in J_i} c_{ij}$ is a polyhedron.*

From Prop. 3, we have that $\overline{\mathcal{S}} = \bigvee_{\ell=0}^{k} C_\ell$ where each $C_\ell = \{\mathbf{t} \in \mathbb{R}^n_{>0} | \mathbf{A}_\ell \cdot \mathbf{t} \trianglelefteq \mathbf{b}_\ell\}$ defines a convex set. In case that the union $\bigvee_{\ell=0}^{k} C_\ell$ is not convex, we use the inclusion-exclusion principle to compute $\Pr^{\mathcal{C}}(\sigma[\overline{\mathcal{S}}])$ as follows:

$$\Pr^{\mathcal{C}}(\sigma[\overline{\mathcal{S}}]) = \sum_{\ell=0}^{k} \Pr^{\mathcal{C}}(\sigma[C_\ell]) - \sum_{i,j:0 \leq i < j \leq k} \Pr^{\mathcal{C}}(\sigma[C_i \wedge C_j]) +$$
$$\sum_{i,j,h:0 \leq i < j < h \leq k} \Pr^{\mathcal{C}}(\sigma[C_i \wedge C_j \wedge C_h]) - \cdots + (-1)^{k-1} \Pr^{\mathcal{C}}(\sigma[C_0 \wedge \cdots \wedge C_k])$$

Remark 4. In our case, the difference between $<$ and $\leq$ in the constraints is marginal, as they would yield the same probability, which can be seen from Eq. (3).

For an index set $L \subseteq \{0, \ldots, k\}$ we write $D = \bigwedge_{\ell \in L} C_\ell$, where C_ℓ defines a polyhedron. By Prop.3, D defines a polyhedron as well. We rewrite $\Pr^{\mathcal{C}}(\sigma[D])$ as:

$$\begin{aligned} \Pr^{\mathcal{C}}(\sigma[D]) &= \prod_{i=0}^{N-1} E(s_i) \cdot P(s_i, s_{i+1}) \cdot \underbrace{\int \cdots \int}_{N}{}_{D} \prod_{i=0}^{N-1} e^{-E(s_i)\tau_i} d\tau_i \\ &= \prod_{i=0}^{N-1} E(s_i) \cdot P(s_i, s_{i+1}) \cdot \underbrace{\int \cdots \int}_{N}{}_{D} e^{-\boldsymbol{E} \cdot \boldsymbol{\tau}} d\boldsymbol{\tau}, \end{aligned}$$

where $\boldsymbol{E} = [E(s_0), \ldots, E(s_{N-1})]$, $\boldsymbol{\tau} = [\tau_0, \ldots, \tau_{N-1}]$ and $\boldsymbol{E} \cdot \boldsymbol{\tau} = \sum_{i=0}^{N-1} E(s_i) \cdot \tau_i$. We use the algorithm of [25] (Sec. 5) to compute efficiently the multidimensional integral $\int \cdots \int_D e^{-\boldsymbol{E} \cdot \boldsymbol{\tau}} d\boldsymbol{\tau}$ based on the Laplace transform. An example of how to compute the integral $\int \cdots \int_D e^{-\boldsymbol{E} \cdot \boldsymbol{\tau}} d\boldsymbol{\tau}$ for a convex set D is given in [15]. The time complexity of solving the multidimensional integral is $\mathcal{O}(n^m)$, where n is the number of constraints and m is the number of variables in D.

Remark 5. Admittedly, it is costly to apply the inclusion-exclusion principle to compute the probabilities. In the worst case, any union of two components is not convex. Notice that efficient algorithms to decide whether the union of two polyhedra is convex there exist; see e.g. [12].

4.3 Main Algorithm and Correctness

We summarize the time-bounded verification algorithm for a CTMC $\mathcal{C}$ against an MTL formula φ in Alg. 3. Recall that Λ is the maximal exit rate appearing in $\mathcal{C}$.

Algorithm 3. Time-bounded verification of a CTMC $\mathcal{C}$ against an MTL formula φ

```
Require: $\mathcal{C}$, $\varphi$, $T$ and $\varepsilon$
Ensure: $\mathrm{Pr}^{\mathcal{C}}_{T,<N}(\varphi)$
  Choose an integer $N \geq \Lambda T e^2 + \ln(\frac{1}{\varepsilon})$
  Transform $\varphi$ into $\widetilde{\varphi}$ and generate NFA $\mathcal{A}_{\widetilde{\varphi}}$ out of $\widetilde{\varphi}$
  Compute the product $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$
  for each discrete path $\sigma$ of $(\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}) \downharpoonleft_1$ of length $n < N$ do
    Generate the family of linear constraints $\mathcal{S}(t_0, \ldots, t_{n-1})$ using Alg. 1 (or Alg. 2)
    Calculate the probability $p$ of $\sigma[\mathcal{S}]$
    $\mathrm{Pr}^{\mathcal{C}}_{T,<N}(\varphi) := \mathrm{Pr}^{\mathcal{C}}_{T,<N}(\varphi) + p$
  end for
  return $\mathrm{Pr}^{\mathcal{C}}_{T,<N}(\varphi)$
```

For the correctness, we first note that the error is bounded by $\mathrm{Pr}^{\mathcal{C}}_{T,\geq N}(\varphi)$, which is in turn bounded by the probability of the set of timed paths with at least N discrete jumps in $[0, T]$. Then Lem. 4 yields the bound, as follows.

Lemma 4. *Given a* CTMC $\mathcal{C}$*, an* MTL *formula* φ*, a time bound* T *and* $N \in \mathbb{N}$

$$\mathrm{Pr}^{\mathcal{C}}_{T}(\varphi) - \mathrm{Pr}^{\mathcal{C}}_{T,<N}(\varphi) \leq \epsilon(T, N).$$

Theorem 3. *Alg. 3 computes* $\mathrm{Pr}^{\mathcal{C}}_{T,<N}(\varphi)$.

5 TA Specifications

In this section, we show how the procedure outlined in the previous section can be adapted to verify TA specifications on CTMCs. Formally, we intend to compute $\mathrm{Pr}^{\mathcal{C}}_{T}(\mathcal{A}) := \mathrm{Pr}^{\mathcal{C}}(\{\rho \in \mathit{Paths}^{\mathcal{C}}_{T} \mid \rho \models_T \mathcal{A}\})$. As in the case of MTL specifications, we bound $\mathrm{Pr}^{\mathcal{C}}_{T}(\mathcal{A})$ by $\mathrm{Pr}^{\mathcal{C}}_{T,<N}(\mathcal{A}) := \mathrm{Pr}^{\mathcal{C}}(\mathit{Paths}^{\mathcal{C}}_{T,<N}(\mathcal{A}))$, such that $\mathrm{Pr}^{\mathcal{C}}_{T}(\mathcal{A}) - \mathrm{Pr}^{\mathcal{C}}_{T,<N}(\mathcal{A}) < \varepsilon$ for $\varepsilon > 0$. The measurability of the set of paths $\mathit{Paths}^{\mathcal{C}}_{T,<N}(\mathcal{A}) := \{\rho \in \mathit{Paths}^{\mathcal{C}}_{T,<N} \mid \rho \models_T \mathcal{A}\}$ can be shown as in [17].

5.1 Constraints Generation

Alur *et. al.* in [5] show how to, given a discrete path π of TA $\mathcal{A}$, construct a graph $\mathcal{G}$ such that $\mathcal{A}$ has a run over π if and only if $\mathcal{G}$ has no negative cost cycle. The graph $\mathcal{G}$ has exactly n nodes and the number of edges of $\mathcal{G}$ depends on the numbers of guards and invariants in $\mathcal{A}$ (see [5] for details). Each edge $e = (i, j)$ (connecting node i to node j) is labeled with a value c such that $c \in \mathcal{H}$ where

$$\mathcal{H} = \{\ldots -2, -1, 0, 1, 2, \ldots\} \cup \{\ldots -2^-, -1^-, 0^-, 1^-, 2^-, \ldots\} \cup \{-\infty, \infty\}$$

The set $\mathcal{H}$ is used to characterize strict and non-strict constraints in $\mathcal{A}$.

For each discrete path σ of the CTMC $\mathcal{C}$ we define $\mathit{\Pi}_\sigma = \{\pi \mid \pi_i \xrightarrow{L(\sigma_i)} \pi_{i+1} \text{ for all } 0 \leq i \leq n-1\}$.

Theorem 4. *Given a discrete path σ of length n, a* TA $\mathcal{A}$ *and a time bound T, we have that $\sigma[t_0, \ldots, t_{n-1}]$ is accepted by $\mathcal{A}$ iff $(t_0, \ldots, t_{n-1}) \in \mathcal{S}$, where $\mathcal{S}$ is returned by Alg. 4.*

Algorithm 4. Constraints generation for a TA

```
Require: A finite discrete path $\sigma$ of length $n > 0$ and a TA $\mathcal{A}$
Ensure: Family of linear constraints $\mathcal{S}$
For the discrete path $\sigma$ compute the set $\mathit{\Pi}_\sigma$
for each $\pi \in \mathit{\Pi}_\sigma$ do
    Generate the graph $\mathcal{G}$
    $\mathcal{S}_\pi := \varnothing$
    for each edge $e(i,j) \in \mathcal{G}$ labeled with $c$ do
        $\mathcal{S}_\pi := \mathcal{S}_\pi \wedge t_i - t_j < c$
    end for
    $\mathcal{S} := \mathcal{S} \vee \Big(\mathcal{S}_\pi \wedge (t_0 + \ldots + t_{n-1} > T \wedge t_0 + \ldots + t_{n-2} < T) \wedge \bigwedge_{0 \leq k < n} t_k > 0\Big)$
end for
return $\mathcal{S}$
```

5.2 Algorithm for TA

Given a timed automaton $\mathcal{A}$ we write $\bar{\mathcal{A}}$ to denote the NFA obtained by removing all the guards, clocks and invariants from $\mathcal{A}$. The product $\mathcal{C} \otimes \bar{\mathcal{A}}$ follows Def. 10. Similarly to Prop. 2, we have that

Proposition 4. *For any* CTMC $\mathcal{C}$ *and NFA $\bar{\mathcal{A}}$, $\mathbb{S}(Paths^{\mathcal{C}}_T(\mathcal{A})) \subseteq \{C_d(\sigma) \mid \sigma \in \Diamond Loc_F \lfloor_1\}$, where Loc_F is the set of final locations in $\mathcal{C} \otimes \bar{\mathcal{A}}$.*

The approximation algorithm for time-bounded verification of a TA specification $\mathcal{A}$ is given in Alg. 5.

Lemma 5. *Given a* CTMC $\mathcal{C}$*, a* TA *specification $\mathcal{A}$, a time bound T and $N \in \mathbb{N}$*

$$\Pr^{\mathcal{C}}_T(\mathcal{A}) - \Pr^{\mathcal{C}}_{T,<N}(\mathcal{A}) \leq \epsilon(T, N).$$

Theorem 5. *Alg. 5 computes $\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A})$.*

Algorithm 5. Time-bounded verification of a TA specification $\mathcal{A}$ against a CTMC $\mathcal{C}$

```
Require: $\mathcal{C}$, $\mathcal{A}$, $T$ and $\varepsilon$
Ensure: $\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A})$
Choose an integer $N \geq \Lambda T e^2 + \ln(\frac{1}{\varepsilon})$
for each discrete path $\sigma$ of $(\mathcal{C} \otimes \bar{\mathcal{A}}) \lfloor_1$ of length $n < N$ do
    Calculate the family of linear constraints $\mathcal{S}(t_0, \ldots, t_{n-1})$ with Algorithm 4
    Calculate the probability $p$ of $\sigma[\mathcal{S}]$
    $\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A}) := \Pr^{\mathcal{C}}_{T,<N}(\mathcal{A}) + p$
end for
return $\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A})$
```

6 Conclusion

In this paper we have studied time-bounded verification of CTMCs against real-time specifications. In particular, we presented effective procedures to approximate the probability of the set of timed paths of CTMCs that satisfy real-time specifications over a time interval of fixed bounded length, arbitrarily closely. For the real-time specifications, we focused on MTL under both the continuous and pointwise semantics, and general timed-automata.

The aim of the current paper is to provide effective approximation algorithms. We leave the precise complexity as future work. Notice that, for MTL, the satisfiability problem over CTMCs is undecidable for continuous semantics [2] while it has non-primitive recursive complexity for pointwise semantics [28]. These results do not carry over directly to CTMCs, as they do not involve nondeterminism. Moreover, we mention that since our algorithms involve computation over reals, it might make more sense to consider different computation models (e.g. the BSS model [13]) and the complexity theory therein, rather than the standard Turing model. Notice that one could also apply discretization to solve the problem. However, it is not clear how the probabilities are preserved in the discretized model.

Recently [26] showed that, under the bounded-variability assumption (BVA), an MTL formula can be transformed into a deterministic timed automaton. Roughly, a timed path satisfies the BVA if there exist Δ and k such that, for *every* interval of the form $[t, t + \Delta]$, the number of discrete jumps is at most k. Clearly, this is related to the bound on discrete jumps in $[0, T]$. However, the BVA is a "global" assumption over $[0, \infty)$, so it does not apply to time-bounded verification. Also, it is not clear for us how to bound the error under this assumption. It would be interesting to investigate whether one could obtain a DTA out of MTL under our assumption of finitely many jumps over $[0, T]$, which could yield an alternative way to solve the problem, based on previous work of two authors [17]. A natural question is how to tackle the traditional (time-unbounded) verification. The scheme introduced in this paper still works. However, one cannot guarantee an approximation to stay within the given error bound ε, which means that the resulting procedure is *not* an approximation algorithm any more. It is also interesting to tackle real-time specifications given as *alternating timed automata* [22] or as TPTL formulas [3,10], as they subsume MTL. We claim that the scheme can be applied in a straightforward way. However, one needs new constraints generation procedures. We leave them as future work.

Acknowledgement. We are grateful to Klaus Dräger, Joost-Pieter Katoen, and anonymous referees for fruitful discussions and constructive comments.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
2. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM 43(1), 116–146 (1996)
3. Alur, R., Henzinger, T.A.: A Really Temporal Logic. J. ACM 41(1), 181–204 (1994)
4. Alur, R., Henzinger, T.A.: Real-time logics: Complexity and expressiveness. In: LICS, pp. 390–401 (1990)

5. Alur, R., Kurshan, R.P., Viswanathan, M.: Membership questions for timed and hybrid automata. In: IEEE Real-Time Systems Symposium, pp. 254–263 (1998)
6. Baier, C., Cloth, L., Haverkort, B.R., Kuntz, M., Siegle, M.: Model checking Markov chains with actions and state labels. IEEE Trans. Software Eng. 33(4), 209–224 (2007)
7. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Software Eng. 29(6), 524–541 (2003)
8. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Performance evaluation and model checking join forces. Commun. ACM 53(9), 76–85 (2010)
9. Baier, C., Hermanns, H., Katoen, J.-P., Haverkort, B.R.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. Theor. Comput. Sci. 345(1), 2–26 (2005)
10. Bouyer, P., Chevalier, F., Markey, N.: On the expressiveness of TPTL and MTL. Inf. Comput. 208(2), 97–116 (2010)
11. Barbot, B., Chen, T., Han, T., Katoen, J.-P., Mereacre, A.: Efficient CTMC model checking of linear real-time objectives. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 128–142. Springer, Heidelberg (2011)
12. Bemporad, A., Fukuda, K., Torrisi, F.D.: Convexity recognition of the union of polyhedra. Comput. Geom. 18(3), 141–154 (2001)
13. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and real computation. Springer, Heidelberg (1998)
14. Bouyer, P.: From Qualitative to Quantitative Analysis of Timed Systems. Mémoire d'habilitation, Université Paris 7, Paris, France (January 2009)
15. Chen, T., Diciolla, M., Kwiatkowska, M., Mereacre, A.: Time-bounded verification of CTMCs against real-time specifications. Tech. Rep. RR-11-06, Department of Computer Science, University of Oxford (2011)
16. Chen, T., Han, T., Katoen, J.-P., Mereacre, A.: Quantitative model checking of continuous-time Markov chains against timed automata specifications. In: LICS, pp. 309–318 (2009)
17. Chen, T., Han, T., Katoen, J.-P., Mereacre, A.: Model checking of continuous-time Markov chains against timed automata specifications. Logical Methods in Computer Science 7(1–2), 1–34 (2011)
18. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM 42(4), 857–907 (1995)
19. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with CSL^{TA}. IEEE Trans. Software Eng. 35(2), 224–240 (2009)
20. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: Time-bounded model checking of infinite-state continuous-time Markov chains. Fundam. Inform. 95(1), 129–155 (2009)
21. Hiriart-Urruty, J., Lemaréchal, C.: Convex Analysis and Minimization Algorithms I.: Fundamentals. Springer, Heidelberg (1994)
22. Jenkins, M., Ouaknine, J., Rabinovich, A., Worrell, J.: Alternating timed automata over bounded time. In: LICS, pp. 60–69. IEEE Computer Society, Los Alamitos (2010)
23. Katoen, J.-P., Zapreev, I.S.: Safe on-the-fly steady-state detection for time-bounded reachability. In: QEST, pp. 301–310 (2006)
24. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems 2(4), 255–299 (1990)
25. Lasserre, J.B., Zeron, E.S.: A Laplace transform algorithm for the volume of a convex polytope. J. ACM 48(6), 1126–1140 (2001)
26. Nickovic, D., Piterman, N.: From MTL to deterministic timed automata. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 152–167. Springer, Heidelberg (2010)
27. Ouaknine, J., Rabinovich, A., Worrell, J.: Time-bounded verification. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 496–510. Springer, Heidelberg (2009)

28. Ouaknine, J., Worrell, J.: On the decidability and complexity of metric temporal logic over finite words. Logical Methods in Computer Science 3(1) (2007)
29. Ouaknine, J., Worrell, J.: Towards a theory of time-bounded verification. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010 Part II. LNCS, vol. 6199, pp. 22–37. Springer, Heidelberg (2010)
30. Roux, O., Rusu, V.: Verifying time-bounded properties for ELECTRE reactive programs with stopwatch automata. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S.S. (eds.) HS 1994 Part II. LNCS, vol. 999, pp. 405–416. Springer, Heidelberg (1995)
31. Schrijver, A.: Theory of linear and integer programming. Wiley-Interscience series in discrete mathematics and optimization. Wiley, Chichester (1999)
32. Sharma, A., Katoen, J.-P.: Weighted lumpability on Markov chains. In: 8th Ershov Informatics Conference. LNCS (2011)
33. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS, pp. 327–338 (1985)
34. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS, pp. 332–344 (1986)

Performance Model Checking Scenario-Aware Dataflow

Bart Theelen[1], Marc Geilen[2], and Jeroen Voeten[1,2]

[1] Embedded Systems Institute
[2] Eindhoven University of Technology, Department of Electrical Engineering
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

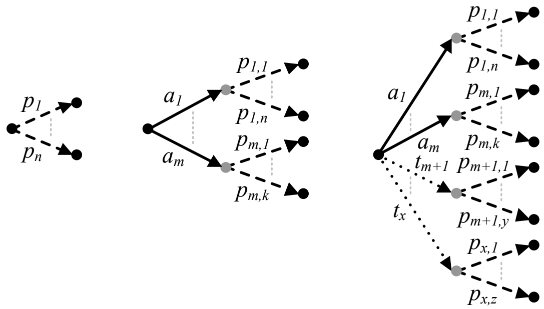
Abstract. Dataflow formalisms are useful for specifying signal processing and streaming applications. To adequately capture the dynamic aspects of modern applications, the formalism of Scenario-Aware Dataflow (SADF) was recently introduced, which allows analysis of worst/best-case and average-case performance across different modes of operation (scenarios). The semantic model of SADF integrates non-deterministic and discrete probabilistic behaviour with generic discrete time distributions. This combination is different from the semantic models underlying contemporary quantitative model checking approaches, which often assume exponentially distributed or continuous time or they lack support for expressing discrete probabilistic behaviour. This paper discusses a model-checking approach for computing quantitative properties of SADF models such as throughput, time-weighted average buffer occupancy and maximum response time. A compositional state-space reduction technique is introduced as well as an efficient implementation of this method that combines model construction with on-the-fly state-space reductions. Strong reductions are possible because of special semantic properties of SADF, which are common to dataflow models. We illustrate this efficiency with several case studies from the multi-media domain.

1 Introduction

Signal processing and streaming applications are often described as a set of tasks, actors or processes with data and control dependencies to exploit the parallel and pipelined execution capabilities of hardware platforms. Modern streaming applications are increasingly dynamic, with large variations in the required resources. Neglecting these variations when evaluating key properties like throughput and buffer occupancy can result in overly pessimistic performance bounds [10], while the average-case behaviour can often not be studied adequately based on the same model. The recently introduced formalism of *Scenario-Aware Dataflow* (SADF) [33] adequately captures dynamism in modern streaming applications using *scenarios*. Such scenarios denote distinct modes of operation (like processing I, P or B frames in MPEG-4 video processing), in which resource requirements can differ substantially [11,25].

This paper presents the techniques underlying the computation of exact worst/best-case and average-case performance numbers for SADF models as implemented in the SDF^3 toolkit [32,29]. Although these techniques are strongly inspired by existing model checking techniques, they are not based on existing model checkers that support quantitative analysis. This is because of semantic differences between the model of SADF and

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 43–59, 2011.



Markov Chain Markov Decision Process Timed Probabilistic System

Fig. 1. Transitions in Probabilistic Automata

the models underlying common quantitative model checkers combined with the diversity of metrics that we want to analyse. The semantics of SADF [34] uses the formalism of *Timed Probabilistic (Labelled Transition) Systems* (TPS) [1]. Like other automata, a TPS describes behaviour in terms of states and transitions. TPS is a variant of probabilistic timed automata in [26] (called Simple Segala Model in [28]), which extend Markov decision processes (MDPs) [5,24] by distinguishing time-less action transitions from transitions for advancing time. Figure 1 shows how MDP alternates non-determinism between actions $a_1, \ldots, a_m$ with probabilistic choices. Actions in TPS are time-less. Time is modelled explicitly using separate transitions, where the labels $t_{m+1} \ldots t_x$ in Figure 1 refer to an *exact* (positive) amount of time (e.g., they do not refer to parameters of exponential distributions). Modelling discrete time distributions can be accomplished by using the two-step approach depicted in Figure 2. It consists of some (internal) action capturing the probabilistic choice of alternative time durations (think of drawing a sample from a discrete distribution), followed by a time transition for each of the possible time durations. In Figure 2, time advances t_i time units with probability p_i for $i = 1, \ldots, n$. This approach is suitable for capturing any discrete time distribution and matches well with the way discrete-event models are commonly implemented. The semantics of SADF in [34] adopts this approach to ease calibration of SADF models with profiling data obtained through static and statistic code analysis.

Several quantitative model checkers exist but using them for SADF is problematic. CADP [12] is a model checking toolbox that supports (amongst others) Interactive Markov Chains (IMC) [13] for quantitative analysis. Although it supports non-deterministic choice between alternative behaviours, IMC itself does not support probabilistic choices and it relies on exponentially distributed time. The probabilistic model checker MRMC [14], which operates on more elementary automata, supports both probabilistic and non-deterministic choices but only in combination with exponentially distributed time. UPPAAL [17] is well-known for its ability to verify qualitative properties of timed systems and the recent extension UPPAAL-PRO adds support for probabilistic choices. Its continuous-time model is again different from the time model of TPS. Nevertheless, TPS can be captured reasonably straightforwardly in UPPAAL-PRO. However, only analysis of maximum probabilistic reachability properties is supported, which is insufficient to compute metrics like throughput and time-weighted

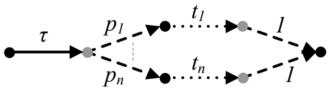


Fig. 2. Generic Discrete Time Distribution

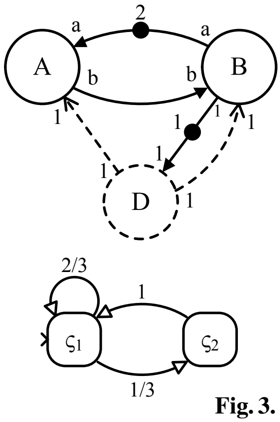
average buffer occupancy. PRISM [15] can verify a wider range of quantitative properties for various probabilistic models including (Priced) Probabilistic Timed Automata. Analysis of reward-based metrics like throughput and buffer occupancy is however not supported for such automata. Furthermore, PRISM lacks features like the concept of urgency in UPPAAL to ease controlling resolution of non-determinism, which is very useful in the context of dataflow formalisms [8,9]. We discuss this aspect in Section 3.

Given the mismatch with existing model checkers, we propose a novel, two-phase approach. In the first phase we construct a reduced, but adequate Markov reward model on which elementary techniques for computing performance numbers can be applied in the second phase, possibly using existing tools. From the basic analysis results, we construct results for more complex performance metrics in a compositional way. This paper focuses on the first phase where we apply several model checking techniques optimized for SADF. These techniques restrain state-space size by exploiting key semantic properties of SADF as well as neglecting events that do not affect a performance result directly. We present how this approach allows for an *on-the-fly* construction of the Markov reward model, which is the key contribution of this paper. A number of case studies from the multi-media domain illustrate the achievable state-space reductions by taking semantic properties and the relevance of events into account.

The remainder of this paper is organised as follows. The next section discusses relevant properties of TPSs defined by SADF semantics. Section 3 presents our performance model checking approach and how it can be implemented efficiently. In Section 4, we present algorithms for computing throughput of SADF models. Section 5 illustrates the efficiency of our approach based on several case studies from the multi-media domain. Conclusions and directions for future research are summarised in Section 6.

2 Scenario-Aware Dataflow

Scenario-Aware Dataflow is an extension of Synchronous Dataflow (SDF) [19] (also known as weighted marked graphs in Petri Net theory), which allows analysis of many correctness and performance properties like absence of deadlock and throughput [27,8]. SADF combines the traditional data-driven behaviour of SDF with state-machine based control behaviour to capture dynamism. Figure 3 shows an illustrative SADF model in the top-left corner. The vertices denote *processes* while the edges are *channels* reflecting (potential) dependencies between those processes. Two types of processes are distinguished. *Kernels* (solid vertices) reflect the data processing part of an application (such as variable length decoding for MPEG-4), whereas *detectors* (dashed vertices) model the control part, responsible for dynamically determining the scenario in which processes operate. The possible orders in which scenarios (ς_1 and ς_2 in Figure 3) occur is captured by state machines. In reality, these state machines coordinate the operation mode based on data-dependent conditions like the type of frame to decode



	Scenario	
Rate	ς_1	ς_2
a	2	0
b	1	0

Process	*Scenario*	E	$\mathbb{P}(E)$
A	ς_1	19	8/11
		51	1/11
		57	2/11
B	ς_1	5	5/19
		17	12/19
		47	2/19
	ς_2	13	1/15
		31	13/15
		63	1/15
D	ς_1	3	4/14
		13	1/14
		29	9/14
	ς_2	7	3/4
		19	1/4

Fig. 3. Example SADF Model

in MPEG-4. SADF abstracts from the actual conditions, taking either a probabilistic or non-deterministic abstraction approach. In case probabilistic information is available on the scenario occurrences, the state machines are (discrete) Markov chains in which case both worst/best-case and average-case analysis becomes possible. Otherwise, the state machines reduce to non-deterministic state machines, which still allows for analysing worst/best-case performance as discussed in [6] for a restricted form of SADF. In this paper, the state machines are considered to be Markov chains in line with [33,34].

A *token* is a unit of information communicated between processes. Such a token can for instance model a frame, line or pixel. The availability of tokens in the (conceptually unbounded) FIFO buffer corresponding to each channel is shown with a dot, labelled with the number of available tokens. Detectors inform other processes about the scenario to operate in by sending them scenario-valued tokens via *control channels*. Control channels are shown as dashed arrows, while solid arrows denote *data channels* (in which data values of tokens have been abstracted). A *production/consumption rate* refers to the number of tokens produced/consumed by a process via a channel each time it *fires*. Rates of 0 are supported to specify absence of data dependencies. The left-hand table in Figure 3 shows that parameterised rates a and b are 0 for scenario ς_2.

The firing of a kernel k starts when one token has become available on (each of) its control input(s). These token(s) determine the scenario, which in turn fixes the parameterised rates. Subsequently, k waits until a number of tokens equal to the consumption rate becomes available on each data input. Then k starts its data processing behaviour, which takes an amount of time given by a sample drawn from the execution time distribution for the active scenario. The right-hand table in Figure 3 shows the possible execution times E and their occurrence probabilities $\mathbb{P}(E)$ for all processes. The firing of k ends by removing a number of tokens, equal to the consumption rate, from each input and producing a number (equal to the production rate) of tokens to each output.

In case a detector d has no control inputs[1], its firing starts with determining its subscenario by making a transition in the associated Markov chain. For detector D, this Markov chain is depicted in the bottom-left corner of Figure 3, where the state names indicate the subscenario to detect. After establishing the subscenario, firing of d

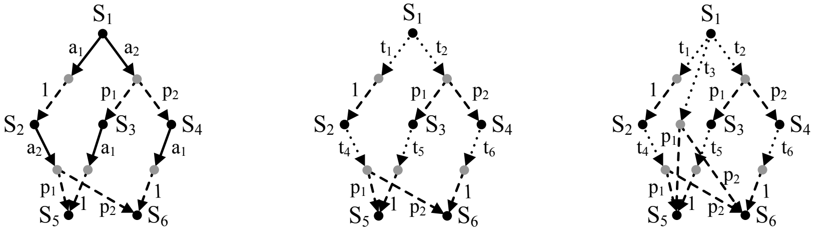
[1] We simplify our explanation in line with [33]. The complete explanation can be found in [34].

continues similarly as a kernel by fixing the parameterised rates. After sufficient tokens have become available on all data inputs, d performs its behaviour which takes an amount of time drawn from the appropriate execution time distribution. Firing of d ends with removing a number, equal to the consumption rate, of tokens from each input and producing a number (equal to the production rate) of tokens to each output, where the tokens written to control channels are valued with the subscenario. Notice that these valued tokens coherently affect the behaviour of kernels A and B in Figure 3. Succinctly capturing such correlations that often exist between dynamic changes in resource requirements for different processes is a key feature of SADF.

2.1 TPS Semantics

Before we discuss the semantics of SADF, we introduce some notation for TPS. Consider a finite set $\mathbb{S}$ of states and let $\mathcal{D}(\mathbb{S})$ denote the set of probability distributions over $\mathbb{S}$, $\mathcal{D}(\mathbb{S}) = \{\boldsymbol{\pi} : \mathbb{S} \to [0,1] \mid \sum_{S \in \mathbb{S}} \boldsymbol{\pi}(S) = 1\}$. A *Timed Probabilistic Systems* (TPS) with initial state $S^* \in \mathbb{S}$ is a transition system $(\mathbb{S}, S^*, \mathcal{A}, \mathbb{A}, \mathcal{T}, \mathbb{T})$ where $\mathcal{A}$ is a finite set of actions, $\mathcal{T}$ denotes the time domain (e.g., the positive reals or integers), while $\mathbb{A}$ and $\mathbb{T}$ are two sets of labelled transition relations. Set $\mathbb{A}$ is a subset of $\mathbb{S} \times \mathcal{A} \times \mathcal{D}(\mathbb{S})$ and defines the action transitions. Relation $S \xrightarrow{a} \boldsymbol{\pi}$ with $\boldsymbol{\pi} \in \mathcal{D}(\mathbb{S})$ holds if action a can be performed from state S, after which the system transits to state T with probability $\boldsymbol{\pi}(T)$. The set $\mathbb{T}$ is a finite subset of $\mathbb{S} \times \mathcal{T} \times \mathcal{D}(\mathbb{S})$ and denotes the time transitions. Relation $S \xrightarrow{t} \boldsymbol{\pi}$ with $\boldsymbol{\pi} \in \mathcal{D}(\mathbb{S})$ holds if from state S the time can advance for a (positive) amount t, after which the system transits to state T with probability $\boldsymbol{\pi}(T)$. If several action and/or time transition relations hold, the choice of which transition is performed is made non-deterministically. Subsequently, a probabilistic choice determines the target state. The visualisation of TPSs in Figures 1 and 2 shows states in $\mathbb{S}$ as black dots. Action transitions are drawn as solid arrows labelled with an action in $\mathcal{A}$, immediately followed (through a grey dot) by a fan-out of dashed arrows representing a distribution in $\mathcal{D}(\mathbb{S})$. Time transitions are depicted as dotted arrows labelled with a time duration in $\mathcal{T}$ and are similarly followed by a probabilistic fan-out.

TPS can be used for defining the semantics of systems in a compositional way. That is, one defines the semantics of each component as TPS, possibly together with



(a) **Action Determinacy** The probabilities on all of the paths from S_1 to each of the states S_5 and S_6 are equal and both actions a_1 and a_2 are performed

(b) **Time Determinacy** The probabilities on all of the paths from state S_1 to each of the states S_5 and S_6 are equal and also $t_1+t_4 = t_2+t_5 = t_2+t_6$

(c) **Time Additivity** The probabilities on all paths from state S_1 to each of the states S_5 and S_6 are equal and $t_3 = t_1 + t_4 = t_2 + t_5 = t_2 + t_6$

Fig. 4. Some Properties of Timed Probabilistic Systems

conditions on actions or advancing time that depend on other components. Parallel composition of the component TPSs resolves such conditions and yields a TPS for the complete system. Figure 4 illustrates a few useful properties [18,22,36] that a TPS may satisfy in general. *Action determinacy* (in a non-probabilistic setting also known as the diamond property) is shown in Figure 4(a). It defines that the net behavioural effect of alternative paths of successive action transitions to a common target state is invariant to how non-deterministic choices between those alternative paths are resolved. Hence, only one (arbitrary) path needs to be explored. Given the discrete time model of TPS, one can consider a similar property for time transitions. *Time determinacy* defines that the net effect in advancing time is invariant to non-deterministic choices, see Figure 4(b). A slightly stronger property is *time additivity* or time continuity [36], which specifies that time transitions can be arbitrarily split into smaller transitions or combined into larger ones. Compared to time determinacy, this requires also the existence of a direct time transition to the target state, see Figure 4(c). Another relevant property that a TPS may satisfy in general is *action persistency*, which indicates that advancing time does not disable any actions that could have been performed.

The semantics of SADF in [34] defines six types of action transitions and one type of time transition. We briefly discuss all these transition types. As described above, each process behaves repetitively according to a fixed pattern. Kernels perform *control*, *start* and *end* actions, whereas detectors follow a pattern of *detect*, *start* and *end* actions. The *control* and *detect* actions determine the (sub)scenario in which a kernel respectively detector is going to operate. They fix the corresponding parameterised rates and execution time distribution when sufficient control tokens are available. The *start* actions happen when sufficient data tokens are available, and *end* actions finalise the firing with consuming and producing the appropriate amounts of tokens from inputs respectively onto outputs. Only *detect* and *start* actions may have a probabilistic fan-out with multiple target states – all other transition types, including time transitions, have a single target state. Together, the probabilistic fan-out of *start* transitions and time transitions follow the pattern of Figure 2 to model discrete time distributions like those exemplified in Figure 3. The probabilistic fan-out of *detect* transitions follow the probabilistic choices of the detector's Markov chain(s). Notice that part of the *control*, *detect* and *start* transitions are conditions on the availability of tokens in the FIFO buffers connected to the inputs of processes. Turning our attention to time transitions, we first remark that there is one global notion of time. Time can advance whenever some process has performed its *start* action, with the additional condition that no *end* actions are enabled. Such *end* action becomes enabled when the remaining execution time for a process has reduced to 0 as a result of advancing time. Time advances (at most) with the amount to enable new *end* actions, unless no actions can become enabled anymore (deadlock).

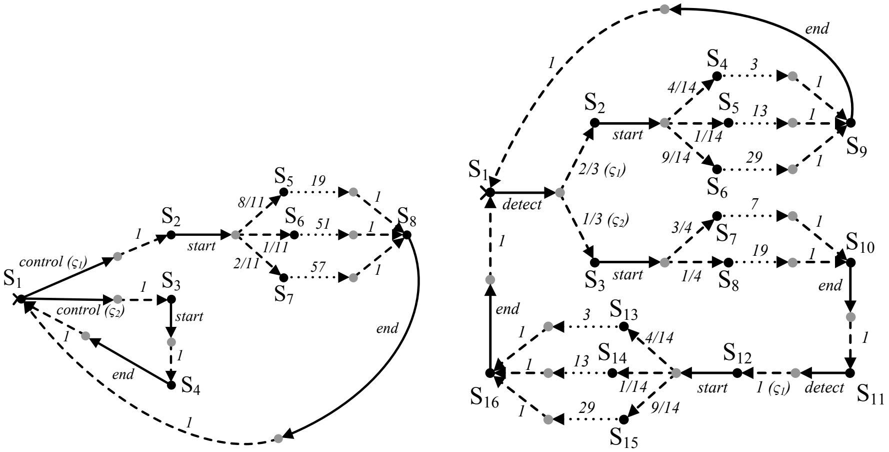
2.2 Semantic Properties of SADF

We can now deduce several important properties of a TPS defined by an SADF model. Our performance model checking approach exploits these to restrain state-space size. To simplify our explanation, we ignore the possibility of deadlocks.

We first discuss time transitions in more detail. For an individual process p, time transitions can only exist after a start action and before the end action. The amount E of advanced time is drawn from an execution time distribution. Observe that the definition of time transitions allows to globally synchronise the advance of time with other processes in a compositional way by being prepared to advance time for p with an arbitrary amount less than E. This requires time additivity, which basically allows splitting time transitions in arbitrarily many smaller time transitions. The composed TPS semantics only explicitly represents the maximum amount of time to pass before an end action is enabled. As a result, at most one time transition can exist from a state and time transitions are always preceded and succeeded by actions. Since time transitions can only be enabled when no process is about to finalise its firing (with an end action), they are succeeded by end actions. Observe that non-determinism between advancing time and performing actions other than end actions is still possible. Consider a state S that is entered after p has performed a start action. A time transition is now enabled for p from all states reachable from S up to states in which some end action is enabled. Although the TPS semantics covers all possible scheduling policies, it is often convenient to assume a specific class of policies that prescribes when actions are to be scheduled for execution. An important class are those policies where actions occur without delays (i.e., all actions are performed before time advances, after which new actions may become enabled again). Such *action urgency* [22] matches with what is known as *self-timed execution* for SDF models [27]. Self-timed execution of SDF models ensures that throughput is maximised [7]. Assuming action urgency may however be disadvantageous for optimising other metrics like latency [9]. Following [33,34,6], we adopt the concept of self-timed execution for SADF. Consistently prioritising actions over advancing time is equivalent to extending the original condition of time transitions in Section 2.1 to one where no action transitions are enabled (i.e., instead of just end actions). This excludes the possibility that both action and time transitions can be enabled from a state.

We now turn our attention to the action transitions. As described above, individual processes exhibit only deterministic behaviour[2] as patterns of control/detect, start, time and end transitions. The actions a process p performs only depend on its own state and the buffer status of channels connected directly to p. Advancing time does not disable any enabled action of p. In fact, actions performed by any process other than p do not disable any enabled action of p, which is stronger than action persistency. This further implies the impossibility of having cycles in the TPS part between any two time transitions in case the system is finite. Deterministic choices may exist between control and detect actions due to mutual exclusive conditions on the values of control tokens, while probabilistic choices may arise from execution time distributions and the Markov chains associated with detectors. This is illustrated in Figure 5 for kernel A and detector D from Figure 3. A can perform two control actions from its initial state S_1 depending on the value of the received control token (ς_1 or ς_2). D determines the subscenario (ς_1 or ς_2) based on the Markov chain in Figure 3 which causes the probabilistic fan-out for the detect actions in Figure 5(b). The parallel composition may yield non-determinism in the composed TPS due to different processes interleaving their independent actions. In other words, any non-determinism in the composed TPS originates from concurrency, and hence it is action determinate [33]. This means that non-determinism between actions can be arbitrarily resolved without affecting the net behaviour of the system.

[2] This is not true if the Markov chains reduce to non-deterministic state-machines as in [6].



(a) TPS for Kernel A

(b) TPS for Detector D

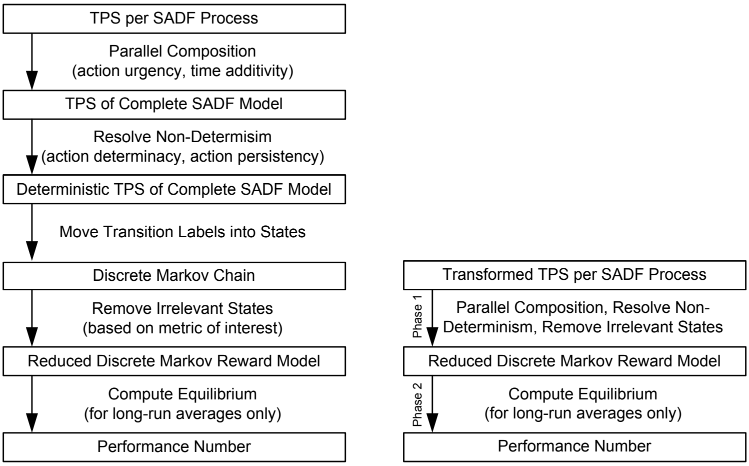
Fig. 5. Semantic Model of Individual Processes (Conditions on Transitions are Omitted)

3 Performance Model Checking

Model checking often exploits properties like action determinacy (diamond property) by means of bisimulation reductions to restrain state-space size. The crux is that redundantly captured behavioural details are removed from the state space. However, such details may be relevant for certain performance metrics [16]. The instantaneous maximum buffer occupancy of SADF channels, for instance, does depend on the order of writing and reading tokens to/from channels. On the other hand, assuming action urgency refers to a class of scheduling policies that (partially) resolves non-determinism. A model checking approach that supports prioritising specific non-deterministic options may avoid constructing the (much larger) state space in which all non-determinism is still available. UPPAAL is an example model checker providing some support for such approach. Finally, certain behaviour may not directly affect a metric, which therefore suggests to consider state spaces that only capture the relevant behaviour. This section presents how our approach exploits these properties to restrain state-space size.

3.1 Strategy for Computing Performance

The proposed strategy, which relies on generic techniques from [35,31], is visualised in Figure 6(a). The idea is to derive a small, but adequate Markov reward model on which elementary techniques for computing concrete performance numbers can be applied. The first step towards a Markov reward model is to construct a TPS of the complete SADF model by parallel composition of the TPSs of individual processes (such as those in Figure 5). This step takes the guards on transitions of the component TPSs into account and uses the property of time additivity and the assumption of action urgency. The resulting composed TPS may include non-deterministic choices between (concurrent) actions and probabilistic choices (from detect and start actions).


TPS per SADF Process
Parallel Composition (action urgency, time additivity)
TPS of Complete SADF Model
Resolve Non-Determisim (action determinacy, action persistency)
Deterministic TPS of Complete SADF Model
Move Transition Labels into States
Discrete Markov Chain
Remove Irrelevant States (based on metric of interest)
Reduced Discrete Markov Reward Model
Compute Equilibrium (for long-run averages only)
Performance Number
Transformed TPS per SADF Process
Phase 1
Parallel Composition, Resolve Non-Determinism, Remove Irrelevant States
Reduced Discrete Markov Reward Model
Phase 2
Compute Equilibrium (for long-run averages only)
Performance Number

(a) Conceptual Analysis Process (b) Practical Implementation

Fig. 6. Performance Model Checking Strategy

The second step is to resolve any remaining non-determinism. Action persistency and action determinacy ensure that time-dependent performance metrics are not affected by the policy for resolving non-determinism (any of the enabled actions may be selected). In [33], this is demonstrated for time-dependent long-run averages like throughput. To compute the instantaneous maximum buffer occupancy, the reservation of buffer space at the start of the producer must be prioritised over the consumption of tokens at the completion of the consumer [34] (i.e., prioritise all start actions over end actions). The resulting TPS can only have probabilistic choices. We denote this TPS by $(\mathbb{S}', S^*, \mathcal{A}, \mathbb{A}', \mathcal{T}, \mathbb{T}')$, where $\mathbb{S}' \subseteq \mathbb{S}$ is the remaining state space and $\mathbb{A}' \subseteq \mathbb{A}$, $\mathbb{T}' \subseteq \mathbb{T}$ are the remaining action and time transitions *after* resolving non-determinism.

The third step is to construct a discrete[3] Markov chain, which is now possible because at most one action or time transition leaves from any state in $\mathbb{S}'$. Performance metrics can be expressed as some combination of reward functions [30] that are evaluated on the obtained Markov chain. We follow the approach of [35] to move the action/time labels of transitions into their (destination) states similarly as in [21,23,3]. The idea is that information on the occurrence of actions and passage of time can be retrieved by reward functions on the states only (i.e., not also on transitions). The state space $\mathcal{S}$ of the Markov chain obtained after the transformation is a subset of $\mathbb{S}' \times (\mathcal{A} \cup \mathcal{T} \cup \{-\})$. The size of $\mathcal{S}$ equals 1 plus the number of transitions with different label/target-state combinations (if $\mathbb{S}'$ is finite) [35]. The interpretation of a state (S, a) in $\mathcal{S}$ is that $S \in \mathbb{S}'$ is entered after having performed action $a \in \mathcal{A}$. State $(S, t) \in \mathcal{S}$ denotes that S is entered after time has advanced with $t \in \mathcal{T}$ time units, while $(S, -) \in \mathcal{S}$ denotes the entrance of S without performing any transition. Observe that only initial state S^* is entered without performing a transition. The one-step transition probabilities of the obtained Markov chain follow straightforwardly from the transitions in $\mathbb{A}'$ and $\mathbb{T}'$.

[3] We deliberately avoid the term discrete-time since there is no relation between the concept of time in TPS and the concept of time in the traditional meaning of discrete-time Markov chains.

The final two steps take the metric of interest into account to compute an exact performance number. We adopt the formalism of *temporal rewards* from [35] to specify metrics. The crux of temporal rewards is that they may not only depend on the current state, like traditional rewards, but also on states visited in the past. We use this ability to express the total amount of time elapsed for a sequence of states, which is an essential component of many time-dependent properties. However, not all states may (directly) contribute to the performance result. We use the state-space reduction technique from [31] to construct a Markov reward model that only includes states in which actions have occurred that *changed* relevant rewards. The actual performance result is then computed from this reduced Markov reward model, after calculating its steady-state or equilibrium distribution in case of long-run average metrics. The next two subsections illustrate how performance metrics can be expressed with temporal rewards and how the reduced Markov reward model is obtained. Section 4 brings the strategy from Figure 6(a) into practice by presenting algorithms performing all steps including constructing the reduced Markov reward model at once, i.e., in an efficient on-the-fly manner.

3.2 Example Performance Metrics

Performance metrics often express properties related to specific actions and/or the advance of time. We illustrate our approach with the more difficult case of long-run averages, since they need to take the equilibrium distribution into account. We aim to exemplify that many long-run averages can be expressed as algebraic combinations of the elementary long-run average [31], which can be evaluated using basic techniques.

Consider the throughput of a process p in an SADF model, which is defined as the long-run average number of firing completions of p per time unit. We use a reward $c : \mathcal{S} \rightarrow \{0, 1\}$ to indicate states in which an action has occurred that directly affects the metric of interest. For the throughput example, $c(U) = 1$ in case p has performed an end action in state $U \in \mathcal{S}$, and $c(U) = 0$ otherwise. States for which c evaluates to 1 are called *relevant*, the others are irrelevant. We further define a temporal reward Δ that gives the sum of time transition labels $t \in \mathcal{T}$ encountered for a realised sequence of states, which restarts the addition each time a relevant state is visited. In other words, Δ denotes the total amount of time elapsed up to visiting the next relevant state.

We now let $\{X_i\}$ denote the stochastic process corresponding to the Markov chain with state space $\mathcal{S}$, where i identifies the i^{th} state visited. The throughput of p can now be expressed as an *event-rate*. For completeness, we give the exact specification in Figure 7, where the limit notation refers to almost sure convergence [2]. It basically divides the number of events (i.e., firing completions of process p) by the average amount of time elapsed between such events. More important is to observe that an event rate resembles the reciprocal of the elementary long-run *sample average* of some reward r, see Figure 7. Many other long-run averages can be expressed as algebraic combinations of sample averages for appropriate definitions of rewards r and c [31]. Examples include the variance in time elapsed between two firing completions of p (a substraction of two sample averages) and the time-weighted average buffer occupancy (a quotient of two sample averages). Hence, given an approach to compute sample averages, many other long-run averages can be computed in a component-wise manner. The next section gives an exact approach to compute sample averages without considering irrelevant states.

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} r(X_i)\cdot c(X_i)}{\sum_{i=1}^{n} c(X_i)} \qquad \lim_{n\to\infty} \frac{\sum_{i=2}^{n} c(X_i)}{\sum_{i=2}^{n} \Delta(X_{i-1})\cdot c(X_i)}$$

sample-average event rate

$$\lim_{n\to\infty} \frac{\sum_{i=2}^{n} r(X_{i-1})\cdot \Delta(X_{i-1})\cdot c(X_i)}{\sum_{i=2}^{n} \Delta(X_{i-1})\cdot c(X_i)}$$

time-weighted average

Fig. 7. Example Generic Forms of Long-Run Averages Comprising Similar Terms

3.3 Metric Dependent State-Space Reduction

The relevance of states indicated by c can be exploited to reduce the state space considerably. Consider an ergodic [4,30] Markov chain $\{X_i\}$ with state space $\mathcal{S}$ for which we need to compute the sample average in Figure 7. By the Ergodic theorem [4,30], it is not difficult to prove [31] that if $\{X_i\}$ has a relevant positive recurrent state, then

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} r(X_i)\cdot c(X_i)}{\sum_{i=1}^{n} c(X_i)} = \frac{\sum_{U\in\mathcal{S}} \pi_U \cdot r(U)\cdot c(U)}{\sum_{U\in\mathcal{S}} \pi_U \cdot c(U)}$$

where π_U is the equilibrium probability of U. Observe that this equation merely depends on rewards earned in relevant states. Hence, it makes sense to investigate whether metrics can be evaluated without considering the irrelevant states at all. Let $\mathcal{S}^c = \{U \in \mathcal{S} \mid c(U) = 1\}$ denote the set of relevant states and define random variable X_i^c as the i^{th} relevant state that is visited (which, for simplicity, we conveniently assume to exist). Now, we can derive some useful theorems, for all of which proofs can be found in [31].

Theorem 1 (Reduction Theorem). *If an ergodic Markov chain $\{X_i\}$ has a relevant positive recurrent state, then $\{X_i^c\}$ is also an ergodic Markov chain.*

Theorem 2 (Preservation of Long-Run Averages). *If an ergodic Markov chain $\{X_i\}$ has a relevant positive recurrent state, then*

$$\frac{\sum_{U\in\mathcal{S}} \pi_U \cdot r(U)\cdot c(U)}{\sum_{U\in\mathcal{S}} \pi_U \cdot c(U)} = \sum_{U\in\mathcal{S}^c} \pi_U^c \cdot r(U)$$

where π^c is the equilibrium distribution of $\{X_i^c\}$.

Hence, the sample average in Figure 7 indeed depends only on rewards r earned in relevant states. Computing it, however, requires equilibrium distribution π^c which relates to the equilibrium distribution π of $\{X_i\}$ as follows.

Theorem 3. *Equilibrium distribution π^c is given by $\pi_U^c = \frac{\pi_U}{\sum_{V\in\mathcal{S}^c} \pi_V}$ for all $U \in \mathcal{S}^c$.*

To avoid storing the complete state space $\mathcal{S}$, we seek a method to determine π^c without first computing π. To present our approach, we consider the matrices of one-step transition probabilities $\mathcal{P}$ and $\mathcal{P}^c$ for the Markov chains $\{X_i\}$ and $\{X_i^c\}$ respectively. To relate $\mathcal{P}^c$ with $\mathcal{P}$, we introduce matrix M. For any $U \in \mathcal{S}$ and $V \in \mathcal{S}^c$, define $M_{U,V}$ as the probability that $\{X_i\}$ makes a sequence of transitions leading to V when departing from U such that any intermediately visited state is irrelevant. Observe that for $U \in \mathcal{S}^c$, $M_{U,V}$ equals the probability $\mathcal{P}^c_{U,V}$ that $\{X^c_i\}$ transfers from U to V.

Theorem 4. *For all $U \in \mathcal{S}$ and $V \in \mathcal{S}^c$, the elements $M_{U,V}$ of matrix M satisfy the system of linear equations given by $M_{U,V} = \mathcal{P}_{U,V} + \sum_{Q \in \mathcal{S} \setminus \mathcal{S}^c} \mathcal{P}_{U,Q} \cdot M_{Q,V}$ which has a unique solution in case the conditions of the reduction theorem are satisfied.*

Consider the computation of $\mathcal{P}^c_{U,V}$ from a given $U \in \mathcal{S}^c$ to a state V in the set $\mathcal{R} \subseteq \mathcal{S}^c$ of all relevant states reachable from U by intermediately visiting irrelevant states only. By constructing the subgraph of $\{X_i\}$ that departs from U and ends in all states of $\mathcal{R}$, the transition probabilities $\mathcal{P}^c_{U,V}$ for each $V \in \mathcal{R}$ can be computed by solving that part of the equations defining M related to this subgraph. Solving the equations is easy, because, as observed in Section 2.2, there are no cycles between states of an SADF model where time has advanced. Computing $\mathcal{P}^c_{U,V}$ for each V therefore boils down to adding the probabilities on the *finite* number of paths between U and V. The subgraph can now be replaced with transitions for $\{X^c_i\}$ between U and each $V \in \mathcal{R}$ with their corresponding probabilities $\mathcal{P}^c_{U,V}$. In this way, only a part of $\{X_i\}$ is constructed at any moment in time while only temporarily storing irrelevant states.

4 Practical Implementation

With the results of Section 3, we propose to construct the reduced Markov reward model in an on-the-fly way. This is visualised as the first phase of our implemented strategy in Figure 6(b), where the second phase computes π^c and the final performance number. We illustrate our approach with the throughput example of Section 3.2, i.e., the event rate in Figure 7. To this end, we introduce some additional notation. Let $\Omega_{U,V}$ denote the set of all paths between two states U and V of $\{X_i\}$ without intermediate visits to relevant states. Such a path refers to a realised sequence of states for $\{X_i\}$ from U to V. We use $\mathbb{P}_\rho$ to denote the probability on a specific path $\rho \in \Omega_{U,V}$, which equals the product of all one-step transition probabilities of $\{X_i\}$ encountered on this path. Observe that if path $\rho \in \Omega_{U,V}$ consists of a single transition, i.e., V is a direct successor of U, then $\mathbb{P}_\rho = \mathcal{P}_{U,V}$. Moreover, for $V \in \mathcal{S}^c$, we have $\sum_{\rho \in \Omega_{U,V}} \mathbb{P}_\rho = M_{U,V}$. We also annotate previously defined temporal reward Δ as Δ_ρ to denote the total amount of time elapsed for a specific path $\rho \in \Omega_{U,V}$. We can now express the expected total amount of time $\overline{\Delta}_U$[4] that elapses until visiting the next relevant state when departing from U as follows

$$\overline{\Delta}_U = \sum_{V \in \mathcal{S}^c} \sum_{\rho \in \Omega_{U,V}} \mathbb{P}_\rho \cdot \Delta_\rho$$

Following from Theorem 2 on preservation of long-run averages, we can now derive for the throughput metric in Section 3.2, i.e., the event rate in Figure 7, that

$$\lim_{n \to \infty} \frac{\sum_{i=2}^{n} c(X_i)}{\sum_{i=2}^{n} \Delta(X_{i-1}) \cdot c(X_i)} = \frac{1}{\sum_{U \in \mathcal{S}^c} \pi^c_U \cdot \overline{\Delta}_U}$$

[4] As a bonus, we like to mention that $\overline{\Delta}_U$ equals the expected response time of a process p in case U is initial state $(S^*, -)$ and c identifies states in which p has performed an end action.

The first phase of our practical implementation in Figure 6(b) constructs $\mathcal{S}^c$ and $\mathcal{P}^c$ to enable computing π^c in the second phase. The first phase also computes $\overline{\Delta}_U$ for all $U \in \mathcal{S}^c$, while the second phase combines the results of all these components into the final performance number.

We present in detail the algorithms for the first phase. It starts from individual TPSs for each process in the SADF model, for which the action/time labels of the transitions have already been shifted into their (destination) states. The initial state $(S^*, -)$ of Markov chain $\{X_i\}$ follows straightforwardly from the initial states of the component TPSs. From $(S^*, -)$, we construct the reduced Markov reward model based on the algorithms in Figure 8. For convenience, we assume that each path $\rho \in \Omega_{U,V}$ from U to some V is stored as a pair $(\mathbb{P}_\rho, \Delta_\rho)$ together with the source state U.

Algorithm *PROGRESS* in Figure 8 constructs all paths from some state $U \in \mathcal{S}$ to relevant states $V \in \mathcal{S}^c$ without intermediate visits to relevant states with a depth-first search. The probabilities $\mathbb{P}_\rho$ and durations Δ_ρ for each of the possible paths ρ are computed while backtracking. After initialising $\mathcal{I}$ and $\mathcal{R}$ to store the newly discovered irrelevant and relevant states respectively, lines 2 through 9 perform a single enabled transition. Line 2 prioritises actions over advancing time conform to the assumption of action urgency. In line 3, an enabled action $a \in \mathcal{A}$ for any process is selected conform to the policy for resolving non-determinism. In case action a is relevant for the performance metric of interest, then *RSTEP* in line 5 determines the immediate next relevant states R, which are added to $\mathcal{R}$. In case any such R is not yet in $\mathcal{S}^c$, it is also added to $\mathcal{S}^c$. In addition, state U is updated to store the paths $\Omega_{U,R}$ for all $R \in \mathcal{R}$ as pairs $(\mathcal{P}_{U,R}, 0)$. On the other hand, if a is irrelevant, then *ISTEP* determines in line 6 the next irrelevant states I, which are added to $\mathcal{I}$. Moreover, U is updated with paths $\Omega_{U,I}$ for all $I \in \mathcal{I}$ as pairs $(\mathcal{P}_{U,I}, 0)$. If in line 3 no actions are enabled but time can advance in line 7 with t units, then *ISTEP* in line 8 adds irrelevant next state I entered after performing the time transition to $\mathcal{I}$ while adding the path to I as the pair $(1, t)$ to U. Line 9 identifies deadlock in case no actions are enabled nor time can advance. Lines 10 through 12 of *PROGRESS* construct the subgraph of $\{X_i\}$ from U until reaching relevant states (which are stored in $\mathcal{R}$). In case a relevant action has been performed, the depth-first search ends. The backtracking procedure in lines 13 through 18 computes the probabilities $\mathbb{P}_\rho$ and durations Δ_ρ for all paths $\rho \in \Omega_{U,V}$ from U to relevant states V, where $\top$ in line 16 denotes a reward that returns the amount of time that has elapsed in each state (which equals 0 if an action was performed). Moreover, the paths of U are properly updated to include only direct paths to the relevant end states V (and thereby discarding all intermediately visited irrelevant states).

Algorithm *CONSTRUCT* in Figure 8 relies on *PROGRESS* to construct the reduced Markov chain $\{X_i^c\}$. After determining the relevant states reachable from a state U, the expected time $\overline{\Delta}_U$ is computed in line 2. For $U \in \mathcal{S}^c$, the probabilities $\mathcal{P}^c_{U,V}$ for all $V \in \mathcal{R}$ are computed in line 5. The recursive construction in line 6 and 7 completes construction of $\{X_i^c\}$ from all relevant states, while the initial call of *CONSTRUCT*$((S^*, -), \varnothing)$ implements the first phase depicted in Figure 6(b).

The algorithms in Figure 8 can easily be adapted to compute performance metrics expressible as a worst/best-case or as (expected) reachability property. The current implementation in SDF^3 [32] for computing various metrics of SADF models uses

```
Algorithm PROGRESS($U, \mathcal{S}^c$)
Input: A source state $U \in \mathcal{S}$ and the current $\mathcal{S}^c$
Output: Set $\mathcal{R} \subseteq \mathcal{S}^c$ of newly discovered relevant states reachable
        from $U$, which are also added to $\mathcal{S}^c$
$\mathcal{I} \leftarrow \varnothing$ and $\mathcal{R} \leftarrow \varnothing$
if actions are enabled
  then select an enabled action $a \in \mathcal{A}$
       if action $a$ is relevant
         then $(\mathcal{R}, U, \mathcal{S}^c) \leftarrow$ RSTEP($U, a, \mathcal{S}^c$)
         else $(\mathcal{I}, U) \leftarrow$ ISTEP($U, a$)
  else if time can advance for $t \in \mathcal{T}$ units
         then $(\mathcal{I}, U) \leftarrow$ ISTEP($U, t$)
  else deadlock detected
for each irrelevant state $I \in \mathcal{I}$
   do $(\mathcal{R}', I, \mathcal{S}^c) \leftarrow$ PROGRESS($I, \mathcal{S}^c$)
      $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}'$
for each irrelevant state $I \in \mathcal{I}$
     for each path $\rho \in \Omega_{U,I}$
        do for each direct successor state $V$ of $I$
               do add path $(\mathbb{P}_\rho \cdot \mathcal{P}_{I,V}, \Delta_\rho + \top(I))$ to $U$
for each irrelevant state $I \in \mathcal{I}$
   do remove all paths $\rho \in \Omega_{U,I}$ from $U$
return $(\mathcal{R}, U, \mathcal{S}^c)$
```

```
Algorithm CONSTRUCT($U, \mathcal{S}^c$)
Input: A source state $U \in \mathcal{S}$ and the current $\mathcal{S}^c$
Output: Updated $\mathcal{S}^c$, $\mathcal{P}^c$ and $\overline{\Delta}$
$(\mathcal{R}, U, \mathcal{S}^c) \leftarrow$ PROGRESS($U, \mathcal{S}^c$)
$\overline{\Delta}_U \leftarrow \sum_{V \in \mathcal{R}} \sum_{\rho \in \Omega_{U,V}} \mathbb{P}_\rho \cdot \Delta_\rho$
if $U \in \mathcal{S}^c$
  then for each $V \in \mathcal{R}$
          do $\mathcal{P}^c_{U,V} \leftarrow \sum_{\rho \in \Omega_{U,V}} \mathbb{P}_\rho$
for each $V \in \mathcal{R}$
   do $(\mathcal{S}^c, \mathcal{P}^c, \overline{\Delta}) \leftarrow$ CONSTRUCT($V, \mathcal{S}^c$)
return $(\mathcal{S}^c, \mathcal{P}^c, \overline{\Delta})$
```

Fig. 8. On-the-fly Construction of the Reduced Markov Reward Model

partially dedicated variants of the algorithms in Figure 8 for doing so, but it also relies on reusing large parts for common terms in different metric types like those in Figure 7. The on-the-fly construction of the reduced Markov reward model has enabled computing the performance of much larger SADF models compared to directly implementing the strategy of Figure 6(a) as we illustrate for several experiments in the next section.

5 Experimental Results

We demonstrate the applicability of our performance model checking approach by computing the throughput for the dynamic applications in the literature listed in Table 1. The examples are ordered in size of their state spaces. The MPEG-4 SP and MP3 examples show increased state spaces when the amount of concurrency by pipelining degree parameter PD increases. All results in Table 1 are obtained using an Intel Centrino 2 based machine at 2.5Ghz running SDF^3 in a virtual machine with 1.5GB of memory. The entries marked – and † in Table 1 denote that it was infeasible to determine the considered aspect either within the available memory or 6 hours of run-time respectively. The $|\mathbb{S}''|$ column presents the size of the composed TPS $(\mathbb{S}, S^*, \mathcal{A}, \mathbb{A}, \mathcal{T}, \mathbb{T})$, where the transition labels have already been shifted into their destination states. In other words, action urgency and time additivity have been applied for parallel composition of the individual TPSs for each process, but non-determinism as a consequence of concurrency has not yet been resolved. Hence, $|\mathbb{S}''|$ is the size of the primary transition system of our approach without taking specific semantic properties of SADF into account. It is clear that concurrency and dynamism easily make construction of this transition system infeasible. The results show that without the possibility of applying any reductions on-the-fly, computing performance of the MPEG-4 SP and MP3 examples would be infeasible.

The $|\mathcal{S}|$ column indicates the size of the Markov chain $\{X_i\}$ obtained after resolution of non-determinism. Resolving non-determinism clearly reduces the state space

Table 1. State-Space Reductions for Throughput Analyses

	Reference	*Remark*	$\lvert\mathbb{S}''\rvert$	$\lvert\mathcal{S}\rvert$	*Process*	$\lvert\mathcal{S}^c\rvert$	*Reduction* [%]	*Run-Time* [s]	*Memory* [MB]
MPEG-4 AVC	[25]		185	183	v_4	18	*90.2*	$\leq$ 0.001	0.272
Running Example	Figure 3		661	375	B	11	*97.1*	0.012	0.384
Channel Equalizer	[20]		2185	296	*cf*	8	*97.3*	0.012	0.672
MPEG-4 SP	[33,32]	PD = 1	–	38440	RC	9	*99.9*	0.8	7.9
MPEG-4 SP	[33,32]	PD = 2	–	483400	RC	576	*99.9*	40.7	16.3
MPEG-4 SP	[33,32]	PD = 3	–	–	RC	8253	–	906.9	94
MP3	[34]	PD = 1	–	–	Write	5	–	26.8	64.6
MP3	[34]	PD = 2	–	–	Write	15	–	624.6	165
MP3	[34]	PD = 3	–	–	Write	15	–	20356	275.5
MP3	[34]	$4 \leq$ PD ≤ 9	–	–	Write	†	–	$>$ 21600	†

State-space sizes: $\lvert\mathbb{S}''\rvert$ is the unreduced size, $\lvert\mathcal{S}\rvert$ is after resolving non-determinism and $\lvert\mathcal{S}^c\rvert$ is after complete reduction

and even allows storing $\{X_i\}$ for some of the MPEG-4 SP examples. The next three columns present the effect of taking the relevance of actions into account for computing the throughput of the listed processes, which are those processes that determine the application's final output. Column $|\mathcal{S}^c|$ gives the size of the reduced Markov reward model $\{X_i^c\}$, which is the number of states that is finally stored at completing the first phase in Figure 6(b). Observe the considerable relative reductions $\frac{|\mathcal{S}|-|\mathcal{S}^c|}{|\mathcal{S}|} \cdot 100\%$ in the eight column that can still be achieved after resolving non-determinism. As we use an on-the-fly implementation to obtain $\{X_i^c\}$, much bigger examples can be analysed without the need to first completely store the primary transition system or $\{X_i\}$, which is the essence of our approach. The one but last column lists the run-times for both phases in Figure 6(b) together, while the last column shows the peak memory usage. Computing the throughput for the full MP3 example (PD = 9) turns out to be infeasible as the case of PD = 4 for this example already requires more than 6 hours of run-time.

6 Conclusions and Outlook

Using state-of-the-art quantitative model checkers for computing exact performance numbers of SADF models is infeasible due to the underlying time model of generic discrete execution time distributions combined with the diversity of the performance metrics of interest. Inspired by generic model checking techniques, this paper proposes a novel approach that exploits various semantic properties of dataflow models, in this case SADF, to counter state-space explosion. The idea to take the relevance of actions into account that directly affect the metric of interest shows the possibility of substantial state-space reductions after resolving non-determinism originating from concurrency, without affecting the final performance number. We proposed an efficient on-the-fly implementation of our approach that does not require storing the complete state space before applying any of the considered reductions. Despite the effectiveness of the proposed approach, several improvements may still be possible when considering state-of-the-art techniques for other semantic models. Concepts like symbolic state space representations and a more component-wise construction of the relevant state space are just a few aspects to investigate. Other directions for future research are to investigate how the metric-dependent Markov chain based reduction technique can be lifted to a proper bisimulation reduction at TPS level to develop a generic TPS-based performance model checker where users can specify any metric of interest in a temporal reward formula.

References

1. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD Thesis. Stanford University (1997)
2. Billingsley, P.: Probability and Measure, 2nd edn. Wiley, Chichester (1995)
3. Chaki, S., Clarke, E.M., Ouaknine, J., Sharygina, N., Sinha, N.: State/Event-Based Software Model Checking. In: Boiten, E.A., Derrick, J., Smith, G.P. (eds.) IFM 2004. LNCS, vol. 2999, pp. 128–147. Springer, Heidelberg (2004)
4. Chung, K.L.: Markov Chains with Stationary Transition Probabilities. Springer, Heidelberg (1967)
5. Derman, C.: Finite State Markovian Decision Processes. Academic Press, London (1970)
6. Geilen, M.C.W., Stuijk, S.: Worst-case Performance Analysis of Synchronous Dataflow Scenarios. In: Proceedings of CODES+ISSS 2010, pp. 125–134 (2010)
7. Govindarajan, R., Gao, G.R.: Rate-Optimal Schedule for Multi-Rate DSP Computations. Journal of VLSI Signal Processing 9, 211–235 (1995)
8. Ghamarian, A.H., Geilen, M.C.W., Stuijk, S., Basten, T., Moonen, A.J.M., Bekooij, M.J.G., Theelen, B.D., Mousavi, M.R.: Throughput Analysis of Synchronous Data Flow Graphs. In: Proceedings of ACSD 2006, pp. 25–34 (2006)
9. Ghamarian, A.H., Stuijk, S., Basten, T., Geilen, M.C.W., Theelen, B.D.: Latency Minimization for Synchronous Data Flow Graphs. In: Proceedings of DSD 2007, pp. 189–196 (2007)
10. Gheorghita, S.V., Stuijk, S., Basten, T., Corporaal, H.: Automatic Scenario Detection for Improved WCET Estimation. In: Proceedings of DAC 2005, pp. 101–104 (2005)
11. Gheorghita, S.V., Basten, T., Corporaal, H.: Application Scenarios in Streaming-Oriented Embedded System Design. In: Proceeedings of SoC 2006, pp. 175–178 (2006)
12. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 372–387. Springer, Heidelberg (2011)
13. Hermanns, H. (ed.): Interactive Markov Chains. LNCS, vol. 2428. Springer, Heidelberg (2002)
14. Katoen, J.-P., Hahn, E.M., Hermanns, H., Jansen, D., Zapreev, I.: The Ins and Outs of the Probabilistic Model Checker MRMC. In: Proceedings of QEST 2009, pp. 167–176 (2009)
15. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of Probabilistic Realtime Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
16. Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.: Performance Analysis of Probabilistic Timed Automata using Digital Clocks. Formal Methods in System Design 29, 33–78 (2006)
17. Larsen, K., Pettersson, P., Yi, W.: UPPAAL in a Nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997)
18. Larsen, K.G., Yi, W.: Time abstracted bisimulation: Implicit specifications and decidability. Information and Communication 134, 75–103 (1997)
19. Lee, E., Messerschmitt, D.: Synchronous Data Flow. IEEE Proceedings 75(9), 1235–1245 (1987)
20. Moonen, A., Bekooij, M., van den Berg, R., van Meerbergen, J.: Practical and Accurate Throughput Analysis with the Cyclo Static Dataflow Model. In: Proceedings of MASCOTS 2007, pp. 238–245. IEEE, Los Alamitos (2007)
21. de Nicola, R., Vaandrager, F.W.: Action versus state based logics for transition systems. In: Guessarian, I. (ed.) LITP 1990. LNCS, vol. 469, pp. 407–419. Springer, Heidelberg (1990)
22. Nicollin, X., Sifakis, J.: An Overview of Synthesis on Timed Process Algebras. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 376–398. Springer, Heidelberg (1992)

23. Obal, W.D., Sanders, W.H.: State-Space Support for Path-Based Reward Variables. Performance Evaluation 35(3/4), 233–251 (1999)
24. Puterman, M.L.: Markov Decesion Processes. Wiley, Chichester (1995)
25. Poplavko, P., Basten, T., van Meerbergen, J.: Execution-Time Prediction for Dynamic Streaming Applications with Task-Level Parallelism. In: Proceedings of DSD 2007, pp. 228–235. IEEE, Los Alamitos (2007)
26. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD Thesis. Massachusetts Institute of Technology (1995)
27. Siram, S., Bhattacharyya, S.S.: Embedded Multiprocessors; Scheduling and Synchronization, 2nd edn. CRC Press, Boca Raton (2009)
28. Sokolova, A.: Coalgebraic Analysis of Probabilistic Systems. PhD Thesis. Eindhoven University of Technology (2005)
29. Stuijk, S., Geilen, M.C.W., Basten, T.: SDF^3: SDF For Free. In: Proceedings of ACSD 2006, pp. 276–278 (2006)
30. Tijms, H.C.: Stochastic Models; An Algorithmic Approach. John Wiley & Sons, Chichester (1994)
31. Theelen, B.D.: Performance Modelling for System-Level Design. Ph.D. Thesis. Eindhoven University of Technology (2004)
32. Theelen, B.D.: A Performance Analysis Tool for Scenario-Aware Streaming Applications. In: Proceedings of QEST 2007 (2007)
33. Theelen, B.D., Geilen, M.C.W., Basten, T., Voeten, J.P.M., Gheorghita, S.V., Stuijk, S.: A Scenario-Aware Data Flow Model for Combined Long-Run Average and Worst-Case Performance Analysis. In: Proceedings of MEMOCODE 2006, pp. 185–194 (2006)
34. Theelen, B.D., Geilen, M.C.W., Stuijk, S., Gheorghita, S.V., Basten, T., Voeten, J.P.M., Ghamarian, A.H.: Scenario-Aware Dataflow. Technical Report ESR-2008-08, Eindhoven University of Technology (July 2008)
35. Voeten, J.P.M.: Performance Evaluation with Temporal Rewards. Performance Evaluation 50(2/3), 189–218 (2002)
36. Yi, W.: Real-time behaviour of asynchronous agents. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 502–520. Springer, Heidelberg (1990)

Probabilistic Real-Time Rewrite Theories and Their Expressive Power

Lucian Bentea and Peter Csaba Ölveczky

Department of Informatics, University of Oslo

Abstract. Unbounded data structures, advanced data types, and/or different forms of communication are often needed to model large and complex probabilistic real-time systems such as wireless sensor networks. Furthermore, it is often natural to model such systems in an object-oriented style, using subclass inheritance and dynamic object and message creation and deletion. To support the above features, we introduce probabilistic real-time rewrite theories (PRTRTs), that extend both real-time rewrite theories and probabilistic rewrite theories, as a rewriting-logic-based formalism for probabilistic real-time systems. We then show that PRTRTs can be seen as a unifying model in which a range of other models for probabilistic real-time systems—including probabilistic timed automata, stochastic automata, deterministic and stochastic Petri nets, as well as two probabilistic timed transition system models with underspecified probability distributions—can naturally be represented.

1 Introduction

In this paper we introduce *probabilistic real-time rewrite theories* (PRTRTs) to support the formal specification of probabilistic real-time systems in rewriting logic [18]. Rewriting logic is a logic for concurrent systems that emphasizes expressiveness and ease of specification over algorithmic decidability of key properties. In rewriting logic, the state space and data types of a system are defined by an algebraic equational specification, and the system's transitions are defined by labeled conditional rewrite rules $l : t \longrightarrow t'$ **if** *cond*, where t and t' are terms that may contain universally quantified mathematical variables. Rewriting logic supports the specification of any computable data type, and distributed systems can be naturally modeled in an object-oriented style, with class inheritance and dynamic creation and deletion of objects and messages. Simulation, reachability analysis, and LTL model checking for rewriting logic is provided by the high-performance Maude tool [8]. (Since properties are in general undecidable, such analyses may not always terminate.)

The Real-Time Maude tool [23] and its real-time rewrite theory formalism [22] extend rewriting logic and Maude to the formal modeling and analysis of real-time systems. Its expressiveness has made it possible to apply the tool to several large applications (see [21]) that are beyond the scope of most model checkers for real-time systems. However, some of those applications, including the LMST

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 60–79, 2011.

and OGDC wireless sensor network algorithms [13,25] and the AER/NCA and NORM multicast protocols [24,17], include probabilistic features—e.g., nodes may exhibit random behavior by design to break symmetries in a network, or the environment may interact with the system in a probabilistic manner—that can only be treated in an *ad hoc* way in Real-Time Maude.

Rewriting logic has also been extended to *probabilistic rewrite theories* to specify probabilistic behaviors [14]. Such theories combine nondeterministic and probabilistic behaviors, and the main idea is that the variables in the righthand side t' of a rewrite rule that do not occur in the lefthand side t are instantiated probabilistically. The VeStA tool [26] can be used for both statistical model checking and estimating numerical values in such theories, and has been used to analyze a DoS resistant TCP/IP protocol [1] and a model of the above mentioned LMST algorithm [13] in which its real-time behavior is treated in an *ad hoc* way.

PRTRTs can be seen as an extension of both real-time rewrite theories and probabilistic rewrite theories. However, PRTRTs are a *proper* extension of probabilistic rewrite theories even when time is ignored. In our case, the new variables in the righthand side of a rule are divided into *nondeterministic* and *probabilistic* variables. Each rewrite rule is equipped with a *family* of probability distributions used to instantiate the probabilistic variables; namely, there is one probability distribution for each substitution of the variables in the lefthand side of the rule *and* each choice of values for the nondeterministic variables. Regarding time, although the duration of/between events may be given probabilistically, time itself does not advance in a probabilistic way in our formalism.

After giving some background into rewriting logic and its probabilistic and real-time extensions (Section 2), we define our formalism and its semantics (Section 3) and the probability of reaching a certain state in a certain time (Section 5). We then demonstrate the expressiveness of PRTRTs—as well as their suitability as a *unifying semantic framework* for probabilistic real-time systems in which different models for such systems can naturally be represented and understood—by showing how a range of well known formal models for probabilistic real-time systems can be seen as PRTRTS (Section 6). Due to lack of space, we refer to the longer technical report [6] for formal proofs of correctness for the representations. To illustrate our formalism, we describe how a simple probabilistic round trip time protocol can be modeled as a PRTRT (Section 4). This system cannot be modeled as an automaton, since the number of messages in the state can grow beyond any bound.

We also explain in the accompanying report [6] how the state-of-the-art OGDC algorithm [30] for wireless sensor networks can be defined as a PRTRT. A main feature of the algorithm is that a sensor node becomes *active* depending on how close it is to an "ideal" position w.r.t. the already active nodes, and that it turns itself off when its sensing area is covered by the sensing areas of other active nodes. The OGDC algorithm therefore requires computing with data types for sensing areas and sophisticated functions including distances, angles, computing overlaps of areas, etc., which seem to be beyond the capability of formalisms that do not support the definition of new data types and advanced functions.

2 Preliminaries

In rewriting logic [18], the static parts of a system (functions, data types, etc.) are defined as an algebraic equational specification, and the transitions of a system are specified by labeled rewrite rules of the form $l:\ t \longrightarrow t'$ **if** *cond*, where t and t' are *terms* constructed by typed variables and function symbols in a type-consistent way, l is a rule label, and *cond* is a (possibly empty) conjunction of equalities, sort memberships, and rewrites. Such a rule specifies a local transition from an instance of the term t to the corresponding instance of the term t', provided that the condition *cond* is satisfied by the substitution instance.

Formally, given a set K of *kinds*, a *many-kinded signature* σ contains a set of function declarations of the form $f : k_1 \dots k_n \to k$, where $n \geq 0$ and $k_1, \dots, k_n, k \in K$. In *membership equational logic* (MEL) [19], each kind k has an associated set of sorts S_k. A *MEL theory* consists of a MEL signature $\Sigma = (K, \sigma, \{S_k \mid k \in K\})$ and a set E of (possibly) conditional equations $(\forall \vec{x})\ \ t = t'$ **if** *cond* and membership axioms $(\forall \vec{x})\ \ t : s$ **if** *cond*, where t and t' are Σ-terms of the same kind k, s is a sort of kind k, *cond* is a conjunction of equalities and sort memberships, and $\vec{x}$ denotes the set of variables in these axioms. We write $vars(t)$ for the set of variables occurring in a term t; if $vars(t) = \emptyset$, then t is called a *ground term*. If $(\Sigma, E \cup A)$ is a MEL theory, where A is a collection of *structural axioms* specifying properties of function symbols, like commutativity, associativity, etc., and E is terminating, confluent and sort-decreasing modulo A, then $Can_{\Sigma,E/A}$ denotes the algebra of fully simplified ground terms, or "normal forms," with respect to the set of axioms E, modulo A. We denote by $[t]_A$ the A-equivalence class of a fully simplified term t. An *E/A-canonical ground substitution* for a set of variables $\vec{x}$ is a function $[\theta]_A : \vec{x} \to Can_{\Sigma,E/A}$ that assigns a fully simplified ground term to each variable in $\vec{x}$. We denote by $CanGSubst_{E/A}(\vec{x})$ the set of all such functions. We use the same notation $[\theta]_A$ for the homomorphic extension of $[\theta]_A$ to Σ-terms.

Definition 1. *A* generalized rewrite theory *[7] is a tuple $\mathcal{R} = (\Sigma, \varphi, E, L, R)$, where Σ is a MEL signature, φ is a function that maps each function symbol $f : k_1 \dots k_n \to k$ in Σ its* frozen argument positions *$\varphi(f) \subseteq \{1, \dots, n\}$, (Σ, E) is a MEL theory, and R is a set of labeled conditional rewrite rules*

$$(\forall \vec{x})\ \ l:\ t \longrightarrow t' \ \ \textbf{if}\ \ cond, \tag{1}$$

where $l \in L$ is a label, t and t' are terms of the same kind, cond is a conjunction of equalities, memberships and rewrites, and $\vec{x} = vars(t) \cup vars(t') \cup vars(cond)$.

Intuitively, if i is a frozen position of a function symbol f, i.e., if $i \in \varphi(f)$, then $f(\dots, t_i, \dots)$ does *not* rewrite to $f(\dots, t'_i, \dots)$ when t_i rewrites to t'_i. A *context* is a Σ-term $\mathbb{C}$ with a single occurrence of a single variable, denoted $\odot$ and called the *hole*. Two contexts $\mathbb{C}$ and $\mathbb{C}'$ are A-equivalent if $A \vdash (\forall \odot)\ \mathbb{C}(\odot) = \mathbb{C}'(\odot)$. A context $f(t_1, \dots, t_n)$ has a hole in a frozen position if the hole occurs in some argument t_i, and either $i \in \varphi(f)$ or t_i has a hole in a frozen position.

Let Ω be a nonempty set. If Ω is countable, a *probability mass function*, or *probability distribution* over Ω is any mapping $p : \Omega \to [0, 1]$ with the property that $\sum_{\omega \in \Omega} p(\omega) = 1$; we denote by $\mathit{Dist}(\Omega)$ the set of all probability distributions over the set Ω. A *cumulative distribution function* (CDF) is a function $\varphi : \mathbb{R} \to [0,1]$ defining the probability $\varphi(x)$ that a real-valued *random variable* is less than $x \in \mathbb{R}$; we write $\mathit{Dist}(\mathbb{R})$ for the set of all CDFs.

In [14] rewrite theories are extended to *probabilistic rewrite theories*. Intuitively, in such theories the righthand side t' of a rewrite rule $l :\ t \longrightarrow t'$ **if** cond may contain variables $\vec{p}$ that do not occur in t. These new variables are assigned values according to a probability distribution taken from a *family* of probability distributions—one for each instance of the variables in t—associated with the rule. Formally, a probabilistic rewrite theory is a pair $(\mathcal{R}, \pi)$, where $\mathcal{R}$ is a rewrite theory[1] and π is a function which assigns to each rule $r \in R$ of the form (1), with $\mathit{vars}(t) = \vec{x}$ and $\mathit{vars}(t') \setminus \mathit{vars}(t) = \vec{p}$, a mapping[2]

$$\pi_r : [\![\mathit{cond}(\vec{x})]\!] \to \mathit{Dist}\left(\mathit{CanGSubst}_{E/A}(\vec{p})\right),$$

where $[\![\mathit{cond}(\vec{x})]\!]$ is the set of all E/A-canonical ground substitutions for $\vec{x}$ that satisfy the condition cond. That is, for each substitution θ of the variables in t which satisfies cond, we get a probability distribution $\pi_r([\theta]_A)$ that defines how the new variables $\vec{p}$ are instantiated. A rewrite rule $r \in R$ of the form (1) with $\mathit{vars}(t') \setminus \mathit{vars}(t) \neq \emptyset$, together with its associated probability distribution function π_r is called a *probabilistic rewrite rule* and is written $l :\ t \longrightarrow t'$ **if** cond **with probability** π_r.

In [22], rewrite theories are extended to real-time systems by *(i)* considering ordinary rewrite rules to be *instantaneous* transitions, and *(ii)* by adding *tick rewrite rules* that model time elapse in a system. Formally, a *real-time rewrite theory* [22] is a pair $(\mathcal{R}, \tau)$ where $\mathcal{R}$ is a generalized rewrite theory and τ is an assignment of a *duration* term τ_l of sort `Time` to rewrite rules of the form $l :$ `{`t`}` $\longrightarrow$ `{`t'`}` **if** cond, where `{_}` encloses the entire state. The term τ_l specifies the amount of time that elapses with the application of the rewrite rule. If $\tau_l \neq 0$, the rule is called a *tick rewrite rule* and is written $l :$ `{`t`}` $\xrightarrow{\tau_l}$ `{`t'`}` **if** cond.

We also use the Maude [8] syntax to specify rewrite rules, so that a conditional tick rule with duration y is written `crl [`l`]: {`t`} => {`t'`} in time` y `if` cond, where the label l may be omitted. In object-oriented Maude specifications [8], the state of the system is a term of sort `Configuration` denoting a multiset of objects and messages, where multiset union is denoted by juxtaposition. Each object is represented as a term `<` $o : c \mid \mathit{att}_1 : \mathit{val}_1,\ \ldots,\ \mathit{att}_n : \mathit{val}_n$ `>`, where o is the object's identifier of sort `Oid`, c is the object's class, and where $\mathit{val}_1, \ldots, \mathit{val}_n$ are the values of the object's attributes $\mathit{att}_1, \ldots, \mathit{att}_n$. For example, the rule

```
rl [l]: m(O, w)  < O : C | a1 : x, a2 : O', a3 : z >   =>
                 < O : C | a1 : x + w, a2 : O', a3 : z >  dly(m'(O'), x) .
```

[1] A (standard) rewrite theory is a generalized rewrite theory with $\varphi(f) = \emptyset$ for all its function symbols f, i.e., a rewrite theory with no frozen operators.

[2] In [14] the definition of probabilistic rewrite theories is based on the more general case of *probability measures* on a σ-algebra over $\mathit{CanGSubst}_{E/A}(\vec{p})$. However, to simplify the exposition, we only consider probability mass functions. Extending our definitions to probability measures is straightforward, as shown in our report [6].

defines a family of transitions in which a message `m`, with parameters `O` and `w`, is read and consumed by an object `O` of class `C`. The transitions change the attribute `a1` of `O` and send a new message `m'(O')` with delay `x`. "Irrelevant" attributes (such as `a3` and the righthand side occurrence of `a2`) need not be mentioned.

3 Probabilistic Real-Time Rewrite Theories

This section defines probabilistic real-time rewrite theories (PRTRTs) and their semantics. PRTRTs extend both probabilistic rewrite theories and real-time rewrite theories to support the formal specification of real-time systems with probabilistic features. The definitions in this section are mostly extensions of similar definitions in [14] for (untimed) probabilistic rewrite theories.

Definition 2. *A* probabilistic real-time rewrite theory *(PRTRT) $\mathcal{R}_{\pi,\tau}$ is a tuple $(\mathcal{R}, \pi, \tau)$, where $\mathcal{R} = (\Sigma, \varphi, E \cup A, L, R)$ is a generalized rewrite theory in which the rules in R have no rewrites in their conditions, $(\mathcal{R}, \tau)$ is a real-time rewrite theory, and π is a function that takes each rewrite rule $r \in R$ of the form (1), with $vars(t) = \vec{x}$, $vars(t') \setminus vars(t) = \vec{y} \uplus \vec{p}$, and assigns to it a mapping*

$$\pi_r : [\![cond(\vec{x} \cup \vec{y})]\!] \to Dist\left(CanGSubst_{E/A}(\vec{p})\right)$$

such that, for each substitution $[\theta]_A \in CanGSubst_{E/A}(\vec{x} \cup \vec{y})$ that satisfies the condition cond, $\pi_r([\theta]_A)$ is a probability mass function over the set of ground substitutions $CanGSubst_{E/A}(\vec{p})$. If r is a tick rule, then $\vec{p} \cap vars(\tau_l) = \emptyset$. Probabilistic tick rewrite rules are written $l : \{t\} \xrightarrow{\tau_l} \{t'\}$ **if** *cond* **with probability** π_r.

Although the duration of/between events may be given probabilistically, time itself does not advance in a probabilistic way. The duration term τ_l therefore does not contain variables that are substituted probabilistically, hence $\vec{p} \cap vars(\tau_l) = \emptyset$. Apart from adding timed behaviors, PRTRTs also extend probabilistic rewrite theories in two ways necessitated by the way tick rules are usually defined:

i) PRTRTs allow new variables that are not assigned a probability distribution in the righthand side of a rewrite rule. The reason is that tick rules—in particular for dense time domains—allow time to advance by *any* amount less than a certain bound; they therefore have a new (non-probabilistic) variable in their righthand sides that defines the duration of the rewrite.
ii) The tick rules involve functions on the state, such as a function defining the effect of time elapse on a state, that are *frozen* operators; we therefore have used *generalized* rewrite theories as the underlying formalism.

Consider the following probabilistic tick rule r (written in an intuitive way):

$$\{f(x)\} \xrightarrow{y} \{g(x, y, z_1, z_2)\} \ \textbf{if} \ \ y \le 10$$
$$\textbf{with probability} \ \ z_1 := \begin{pmatrix} h(x,y) & f(y) \\ 2/3 & 1/3 \end{pmatrix} \ \textbf{and} \ \ z_2 := \begin{pmatrix} 0 & 1 \\ 1/2 & 1/2 \end{pmatrix}.$$

The righthand side term `{`$g(x, y, z_1, z_2)$`}` contains variables y, z_1 and z_2 that do not occur in the rule's lefthand side `{`$f(x)$`}`. Let `{`$f(t)$`}` be the state of the system when the rule is applied. The variable y is then instantiated *nondeterministically* with any value t' less than or equal to 10. The variables z_1 and z_2 are then instantiated probabilistically, where z_1 is assigned the value $[h(t, t')]_A$ with probability 2/3 and the value $[f(t')]_A$ with probability 1/3. Formally, the mapping $\pi_r : \mathit{CanGSubst}_{E/A}(\{x, y\}) \to \mathit{CanGSubst}_{E/A}(\{z_1, z_2\})$ associated to the above rule r is given by $\pi_r([\theta]_A)(\{z_1 \mapsto [h(\theta(x), \theta(y))]_A,\ z_2 \mapsto i\}) = 1/3$ and $\pi_r([\theta]_A)(\{z_1 \mapsto [f(\theta(y))]_A,\ z_2 \mapsto i\}) = 1/6$, for $i \in \{[0]_A, [1]_A\}$.

Let $\mathcal{R}_{\pi,\tau} = (\Sigma, \varphi, E \cup A, L, R, \pi, \tau)$ be a PRTRT. Intuitively, an R/A-match contains the complete information on how and in which context the current system state is matched against a particular rewrite rule in the specification of that system. We extend the definition of R/A-matches in [14] as follows:

Definition 3. *Given a fully simplified term* $[u]_A \in \mathit{Can}_{\Sigma,E/A}$*, its* generalized R/A-matches *are triples* $([\mathbb{C}]_A, r, [\theta]_A)$ *where* $\mathbb{C}$ *is a context whose hole is not in a frozen position,* $r \in R$ *is a rewrite rule,* $[\theta]_A \in \mathit{CanGSubst}_{E/A}(\vec{x} \cup \vec{y})$ *is a substitution such that* $E \cup A \vdash \theta(cond)$*, and* $[u]_A = [\mathbb{C}(\odot \leftarrow \theta(t))]_A$ *is the* A*-equivalence class of the term obtained by applying the substitution* θ *to* t *and placing the result into* $\mathbb{C}$*.*

The definition of a single (instantaneous and tick) transition of a PRTRT describes how the system state evolves when applying a rewrite rule to it:

Definition 4. *Given terms* $[u]_A, [v]_A \in \mathit{Can}_{\Sigma,E/A}$*, an* E/A-canonical one-step rewrite *from* $[u]_A$ *to* $[v]_A$ *is a labelled transition* $[u]_A \xrightarrow[\tau]{([\mathbb{C}]_A,\, r,\, [\theta]_A,\, [\rho]_A)} [v]_A$*, where* $([\mathbb{C}]_A, r, [\theta]_A)$ *is a generalized* R/A*-match for* $[u]_A$ *selected nondeterministically,* $[\rho]_A \in \mathit{CanGSubst}_{E/A}(\vec{p})$ *is a substitution selected with probability* $\pi_r([\theta]_A)([\rho]_A)$*, the duration* τ *is* 0 *if* r *is not a tick rule and is* $\theta(\tau_l)$ *otherwise, and* $[v]_A = [\mathbb{C}(\odot \leftarrow t'(\theta(\vec{x}, \vec{y}), \rho(\vec{p})))]_A$ *is the result of the one-step rewrite.*

4 Example: A Simple Round Trip Time Protocol

To illustrate our formalism, we specify in an object-oriented way a simple *round trip time* (RTT) protocol that computes the time it takes for a message to go from one node to another, and back, and where the message transmission time follows a probability distribution that depends on the distance between the nodes.

The initiator object `O` starts the protocol by sending an `rttReq` message to its neighbor `O'`, with a time stamp `T` which is the current value of `O`'s local clock (rule `start`). When `O'` receives this message, it immediately sends back a reply to `O` with the original time stamp with probability 3/4 and ignores the request with probability 1/4 (rule `rttResp`). When the initiator `O` receives the reply, it computes its RTT value w.r.t. `O'` by subtracting the original time stamp `T` from its current clock value `T'` (rule `treatResp`). However, if the message takes so long that `T'` $-$ `T` $\geq$ `MAX_RTT`, then it is just ignored (rule `ignoreOld`). The initiator uses a retransmission timer to start a new round of the protocol every MAX_RTT time units until it has computed a good RTT value. When the timer expires, O sends another RTT request to O' (rule tryAgain) with probability $1/(\mathtt{N}+1)$, which decreases with the number N of unresolved RTT requests of O. We represent each node by an object

< o : Node | nbr : o', rtt : r, clock : t, timer : ti, tries : n >

where o is the node's identifier, o' is the neighbor to which o wants to compute its round trip time, r is the value of the round trip time, if computed, or INF otherwise, t is the current value of the node's clock, ti is its current timer value, which has the value INF if the timer is switched off, and n is the number of unsuccessful attempts that o has made to compute the RTT. Messages have the form findRtt(o), which triggers a run of the RTT protocol for node o, rttReq(o', o, t), which sends a request from node o to node o' with t the time stamp, and rttResp(o, o', t), which sends a reply message from node o' to node o with the original time stamp t. We assume that a function dist that computes the distance dist(o, o') between two nodes is defined. In the rules start, rttResp and tryAgain we specify the transmission delay of the rttReq and rttResp messages as a variable D which is probabilistically substituted according to a probability distribution F(x) that mimics a truncated normal distribution $\mathcal{N}(\mu, \sigma^2)$ [12] with minimum value MIN_DELAY, and depends on the distance x between o and o', where μ and σ are positive constants representing the average and the standard deviation of the transmission delay, respectively.

The following instantaneous rewrite rules describe our simple RTT protocol.

```
vars O O': Oid. vars T T' D: Time. var N: Nat. var B: Bool. var CF: Configuration.

rl [start] :
   findRtt(O)  < O : Node | clock : T, nbr : O' >
 =>  < O : Node | timer : MAX_RTT >  dly(rttReq(O', O, T), D)
     with probability D := F(dist(O, O')) .

rl [rttResp] :
   rttReq(O, O', T)  < O : Node | >
 =>  if B then < O : Node | >  dly(rttResp(O', O, T), D)  else < O : Node | >  fi
                             / true  false \
     with probability B :=  |              |  and D := F(dist(O, O')) .
                             \  3/4   1/4  /

crl [treatResp] :
    rttResp(O, O', T)  < O : Node | clock : T' >
 =>  < O : Node | rtt : T' - T, timer : INF >  if T' - T < MAX_RTT .

crl [ignoreOld] :
    rttResp(O, O', T)  < O : Node | clock : T' >  =>  < O : Node | >  if T' - T >= MAX_RTT.

rl [tryAgain] :
    < O : Node | timer : 0, clock : T, nbr : O', tries : N >
 =>  if B then  < O : Node | timer : MAX_RTT, tries : N + 1 >  dly(rttReq(O, O', T), D)
     else < O : Node | timer : MAX_RTT >  fi
                             /   true     false  \
     with probability B :=  |                     |  and D := F(dist(O, O')) .
                             \ 1/(N+1)  N/(N+1)  /
```

Time elapse is modeled by the tick rule

```
crl [tick] : {CF}  =>  {delta(CF, T)} in time T  if T <= mte(CF) .
```

where `delta` is a frozen function that specifies the effect of time elapse on the system by decreasing the timers and increasing the clock values of each node. The frozen function `mte` gives the maximum amount of time that can elapse before a node must perform an instantaneous transition. More precisely, time cannot advance past the expiration of a timer or the moment when a message arrives. See [6] for their formal definition.

It is worth noticing that the number of messages in the state can grow beyond any bound, since: *i)* the message delays could be arbitrarily large (with non-zero probability), and *ii)* the initiator node will periodically send requests until it receives a good RTT value. Therefore, even this simple protocol seems to be beyond the scope of automaton-based formalisms.

5 Reachability Probabilities in PRTRTs

In this section we define the probability of reaching a given state in a certain time in a PRTRT. PRTRTs combine probabilistic and nondeterministic behaviors, and we must therefore assign "probabilities" also to the nondeterministic choices to define the probability of reaching a state t_2 from a state t_1 in time τ. This is done by "adversaries," so that the probability of reaching t_2 in time τ is defined relative to a given adversary.

A *computation* of a PRTRT $\mathcal{R}_{\pi,\tau}$ is an infinite sequence of E/A-canonical rewrite steps, with zero-time self-loops from deadlock states. Formally, a computation of $\mathcal{R}_{\pi,\tau}$ is an *infinite* sequence $\Pi = [u_1]_A \xrightarrow[\tau_1]{\alpha_1} [u_2]_A \xrightarrow[\tau_2]{\alpha_2} \cdots$ where $[u_i]_A \xrightarrow[\tau_i]{\alpha_i} [u_{i+1}]_A$ is either a E/A-canonical one-step rewrite, or $[u_i]_A$ cannot be rewritten using the rules in $\mathcal{R}_{\pi,\tau}$, in which case $[u_i]_A = [u_{i+1}]_A$, $\alpha_i = !$, and $\tau_i = 0$, where '!' is a new label. To each computation Π we associate the *infinite timed computation path* obtained by removing the labels above the transition arrows. A *finite timed computation path* is a prefix of an infinite timed computation path. We denote by $\Pi_n = [u_1]_A \xrightarrow[\tau_1]{} \ldots \xrightarrow[\tau_{n-1}]{} [u_n]_A$ a finite timed computation path. The nondeterministic choices of the generalized R/A-matches $([\mathbb{C}]_A, r, [\theta]_A)$ for a state $[u_k]_A$ prohibit us from defining the probability of reaching $[u_n]_A$ in a certain time. Therefore, all nondeterministic choices must be resolved by means of an *adversary*, which extends the notion of adversary in [14] as follows:

Definition 5. *An R/A-*match adversary *of a PRTRT $\mathcal{R}_{\pi,\tau}$ is a function $\mathcal{A}$ that maps each finite timed computation path Π_n of $\mathcal{R}_{\pi,\tau}$, ending with a term $[u_n]_A$, to a probability distribution on the set of generalized R/A-matches for $[u_n]_A$.*

Given an R/A-match adversary $\mathcal{A}$, the following formula gives the probability of performing a rewrite from a "non-deadlock" term $[u_n]_A$ to a term $[u']_A$ in one step in time τ, provided that $[u_n]_A$ is obtained via Π_n,

$$\mathbb{P}_{\mathcal{A}}\left([u_n]_A \xrightarrow[\tau]{} [u']_A \mid \Pi_n\right) = \sum \Big[\mathcal{A}(\Pi_n)\left([\mathbb{C}]_A, r, [\theta]_A\right) \cdot \pi_r([\theta]_A)\left([\rho]_A\right)\Big],$$

where the sum ranges over all $[u_n]_A \xrightarrow[\tau]{([\mathbb{C}]_A,\, r,\, [\theta]_A,\, [\rho]_A)} [u']_A$. If $[u_n]_A$ is a deadlock state then $\mathbb{P}_{\mathcal{A}}\left([u_n]_A \xrightarrow[\tau]{} [u']_A \mid \Pi_n\right) = 1$ if and only if $[u']_A = [u_n]_A$ and $\tau = 0$, and is 0 otherwise. Also, the probability of the finite timed computation path Π_n to occur in a PRTRT is given by:

$$\mathbb{P}_{\mathcal{A}}\left(\Pi_n\right) = \prod_{i=1}^{n-1} \mathbb{P}_{\mathcal{A}}\left([u_i]_A \xrightarrow[\tau_i]{} [u_{i+1}]_A \,\middle|\, [u_1]_A \xrightarrow[\tau_1]{} \cdots \xrightarrow[\tau_{i-1}]{} [u_i]_A\right).$$

The probability of reaching state $[u']_A$ from state $[u]_A$ in time τ is given by $\sum_{\Pi} \mathbb{P}_{\mathcal{A}}(\Pi)$, where Π ranges over all finite timed computation paths Π_n with $[u_n]_A = [u']_A$ and $\sum_{i=1}^{n-1} \tau_i = \tau$, and such that there is no $j < n$ with $[u_j]_A = [u']_A$ and $\sum_{i=1}^{j-1} \tau_i = \tau$.

6 The Expressive Power of PRTRTs

In this section we show the expressiveness of PRTRTs—and its suitability as a unifying semantic framework for probabilistic real-time systems in which different models of such systems can be represented—by explaining how a range of models of probabilistic real-time systems can naturally be seen as PRTRTs. More details about the mappings and their correctness proofs are given in [6].

Since probabilistic rewrite theories are a proper subclass of PRTRTs, any probabilistic rewrite theory can also be represented as a PRTRT. In [14] mappings are provided from probabilistic nondeterministic systems, generalized semi-Markov processes, and continuous-time Markov chains into probabilistic rewrite theories. That paper also claims that the same method can be used for representing the PEPA [11] language and various Petri net formalisms, such as stochastic reward nets, generalized stochastic Petri nets [3], and stochastic Petri nets with generally distributed firing times, as probabilistic rewrite theories.

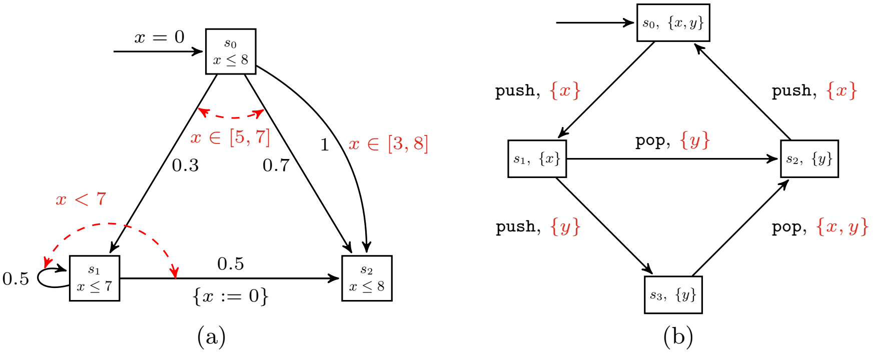
6.1 Probabilistic Timed Automata

The *probabilistic timed automaton* (PTA) model [27] combines nondeterministic and probabilistic behaviors, and extends timed automata [5] by allowing a probabilistic choice of both the *next state* and the *set of clocks* to be reset in a "transition." PTA are supported by the probabilistic model checker PRISM [15].

A *clock* is a variable ranging over the real numbers that increases its value according to the elapsed time. A *zone* of a set of clocks $\mathcal{X}$ is a convex subset of $\mathbb{R}^{|\mathcal{X}|}$ defined by a conjunction of constraints over $\mathcal{X}$. Let $Z_{\mathcal{X}}$ be the set of all zones of $\mathcal{X}$. A PTA is then a tuple $(S, s_0, \mathcal{X}, \mathrm{inv}, \mathrm{prob}, \{\tau_s\}_{s \in S})$, where: S is a finite set of *states* with $s_0 \in S$ the *start* state; $\mathcal{X}$ is a finite set of *clocks*; $\mathrm{inv} : S \to Z_X$ is a function that assigns an *invariant* to each state; $\mathrm{prob} : S \to \mathcal{P}(\mathit{Dist}\,(S \times \mathcal{P}(\mathcal{X})))$ is a function that *assigns a set of probability distributions* on $S \times \mathcal{P}(\mathcal{X})$ to each state; and $\{\tau_s\}_{s \in S}$ is a family of functions where, for each $s \in S$, $\tau_s : \mathrm{prob}(s) \to Z_{\mathcal{X}}$ assigns an *enabling condition* to each $p \in \mathrm{prob}(s)$.

A PTA in state s may *nondeterministically* select any enabled probability distribution p in $\mathrm{prob}(s)$. The probability that the automaton then makes a transition to state s' and resets all the clocks $X \subseteq \mathcal{X}$ to 0 is $p(s', X)$. Following [16,27], we assume that the enabling condition $\tau_s(p)$ implies the invariant of all possible successor states s' with $p(s', X) > 0$, *after* the clocks in X are reset.

Figure 1 (a) shows a PTA that starts in state s_0 with its clock x initialized to 0. When $x \in [5, 7]$ in state s_0 the automaton can nondeterministically choose between the two probability distributions $\pi_1 = \begin{pmatrix} (s_1, \emptyset) & (s_2, \emptyset) \\ 0.3 & 0.7 \end{pmatrix}$ and $\pi_2 = \begin{pmatrix} (s_2, \emptyset) \\ 1 \end{pmatrix}$, and when $x \in [3, 8] \setminus [5, 7]$ it can only take π_2. A probabilistic choice is then made according to the selected probability distribution, and the corresponding transition is performed. Likewise, the probabilistic choice from s_1 is made by sampling from the probability distribution $\pi_3 = \begin{pmatrix} (s_1, \emptyset) & (s_2, \{x\}) \\ 0.5 & 0.5 \end{pmatrix}$; if the automaton makes a transition to state s_2, which happens with probability 0.5, then x is reset to 0.



(a)

(b)

Fig. 1. (a) A probabilistic timed automaton (b) A stochastic automaton

A PTA $A = (S, s_0, \mathcal{X}, \mathrm{inv}, \mathrm{prob}, \langle \tau_s \rangle_{s \in S})$ with $\mathcal{X} = \{x_1, \ldots, x_n\}$ is represented as a PRTRT $\Psi_{PTA}(A)$ as follows (see [6] for more details). A "timed state" of A is represented as a term `{`$s, r_1, r_2, \ldots, r_n$`}`, with r_i denoting the current value of clock x_i. To each state $s \in S$ and each probability distribution $\pi : S \times \mathcal{P}(\mathcal{X}) \to [0, 1]$ in $\mathrm{prob}(s)$, we associate a probabilistic rewrite rule

```
crl [π]: {s, y1, ..., yn} => σ if (y1, ..., yn) ∈ τs(π) with probability σ := Γs(π)
```

where σ and the y_i are variables, and $\Gamma_s(\pi) : \mathit{CanGSubst}_{E/A}(\sigma) \to [0, 1]$ is a probability distribution over the set of E/A-canonical ground substitutions for σ defined by $\Gamma_s(\pi)(\sigma \mapsto$ `{`$s', t_1, \ldots, t_n$`}`$) = \pi(s', X)$ for all $s' \in S$ and all $X \subseteq \mathcal{X}$, where t_j is 0 if $x_j \in X$ and y_j otherwise. A tick rule

```
crl [tick s]:
    {s, y1, ..., yn} => {s, y1 + y, ..., yn + y} in time y if (y1 + y, ..., yn + y) ∈ inv(s)
```

models time elapse for each state $s \in S$. Since $\tau_s(\pi)$ and $\mathrm{inv}(s)$ are zones defined by conjunctions of inequality constraints over the clock values, the set memberships $(y_1, y_2, \dots, y_n) \in \tau_s(\pi)$ and $(y_1 + y, y_2 + y, \dots, y_n + y) \in \mathrm{inv}(s)$ are actually translated into standard inequalities in $\Psi_{PTA}(G)$, as shown below.

The probabilistic timed automaton in Fig. 1 (a) is therefore represented by a PRTRT containing the following set of conditional tick rules

```
crl [tick s0]: {s0, x} => {s0, x + y} in time y if x + y <= 8 .
crl [tick s1]: {s1, x} => {s1, x + y} in time y if x + y <= 7 .
crl [tick s2]: {s2, x} => {s2, x + y} in time y if x + y <= 8 .
```

as well as the following instantaneous probabilistic rewrite rules

$$\texttt{crl [}\pi_1\texttt{]: \{}s_0, x\texttt{\} => }\sigma\texttt{ if }x\texttt{ >= 5 and }x\texttt{ <= 7 with probability }\sigma \texttt{ := } \begin{pmatrix} \texttt{\{}s_1,\ x\texttt{\}} & \texttt{\{}s_2,\ x\texttt{\}} \\ \texttt{0.3} & \texttt{0.7} \end{pmatrix} \texttt{ .}$$

$$\texttt{crl [}\pi_2\texttt{]: \{}s_0, x\texttt{\} => }\sigma\texttt{ if }x\texttt{ >= 3 and }x\texttt{ <= 8 with probability }\sigma \texttt{ := } \begin{pmatrix} \texttt{\{}s_2,\ x\texttt{\}} \\ \texttt{1.0} \end{pmatrix} \texttt{ .}$$

$$\texttt{crl [}\pi_3\texttt{]: \{}s_1, x\texttt{\} => }\sigma\texttt{ if }x\texttt{ < 7 with probability }\sigma \texttt{ := } \begin{pmatrix} \texttt{\{}s_1,\ x\texttt{\}} & \texttt{\{}s_2,\ \texttt{0}\texttt{\}} \\ \texttt{0.5} & \texttt{0.5} \end{pmatrix} \texttt{ .}$$

where σ, x and y are variables and the initial state is given by the term $\texttt{\{}s_0, \texttt{0}\texttt{\}}$.

6.2 Stochastic Automata

A *stochastic automaton* (SA) [9] is an automaton where the transitions have the form $s \xrightarrow{a,X} s'$, with a an action and X a set of *timers*. A transition is enabled when all the timers in X have expired, and time cannot advance when there is an enabled transition. An SA may also have nondeterministic behaviors, since multiple transitions may become enabled at the same time. As the result of taking the transition $s \xrightarrow{a,X} s'$, each timer x which should be reset when arriving at s' is assigned a value sampled from its cumulative distribution function $F(x)$.

Figure 1 (b) shows an SA. Each state has a set of timers that have to be set when arriving at that state. The SA starts in state s_0 and uses the CDFs $F(x)$ and $F(y)$ to assign initial values to the timers x and y. The automaton makes a (`push`) transition to state s_1 as soon as the timer x expires. In the new state s_1, the timer x is assigned a new value sampled from the CDF $F(x)$. At this point, the SA makes a nondeterministic choice between the two possible transitions when the timer y expires, to s_2 with a `pop` action, or to s_3 with a `push` action.

Let $\mathcal{P}_{\mathrm{fin}}(X)$ denote the set of *finite* subsets of a set X. A stochastic automaton is a tuple $(S, s_0, \mathcal{X}, \mathrm{Act}, \longrightarrow, \kappa, F)$ where: S is a set of *states* and $s_0 \in S$ is the *initial state*; $\mathcal{X}$ is a set of *timers*; Act is a set of *actions*; $\longrightarrow \subseteq S \times (\mathrm{Act} \times \mathcal{P}_{\mathrm{fin}}(\mathcal{X})) \times S$ is the set of *transitions*, where we write $s \xrightarrow{a,X} s'$ iff $(s, a, X, s') \in \longrightarrow$; the function $\kappa : S \to \mathcal{P}_{\mathrm{fin}}(\mathcal{X})$ is the *clock setting function*; and $F : \mathcal{X} \to \mathrm{Dist}(\mathbb{R})$ assigns a CDF $F(x)$ to each timer $x \in \mathcal{X}$, with $F(x)(t) = 0$ if $t < 0$.

We outline the PRTRT representation $\Psi_{SA}(A)$ of a finitary SA $A = (S, s_0, \mathcal{X}, \mathrm{Act}, \longrightarrow, \kappa, F)$, where $\mathcal{X}$ is a finite set $\{x_1, \dots, x_n\}$. The “timed state” of A is represented by a term $\texttt{\{}s, r_1, r_2, \dots, r_n\texttt{\}}$, where s is a constant denoting the current state of the SA, and r_i is the current value of the timer x_i. The following probabilistic rewrite rule randomly selects the initial timer values:

```
rl [init]: init => {$s_0$, $y_1, \ldots, y_n$} with probability $y_1$ := $F_1$ and ... and $y_n$ := $F_n$ .
```

where `init` is a constant and F_i mimics the CDF $F(x_i)$ of clock x_i. Each transition $s \xrightarrow{a,X} s'$ of the SA is translated to a labeled probabilistic rewrite rule

```
rl [$a$]: {$s, r_1, \ldots, r_n$} => {$s', r'_1, \ldots, r'_n$} with probability $r'_{j_1}$ := $F_{j_1}$ and ... and $r'_{j_l}$ := $F_{j_l}$.
```

where r_i is 0 if $x_i \in X$ and is a variable y_i otherwise, $\kappa(s') = \{x_{j_1}, x_{j_2}, \ldots, x_{j_l}\}$, and $r'_i = r_i$ if $x_i \notin \kappa(s')$. Time elapse is modeled by the following tick rule which can advance time until the next timer expires if no transition is enabled:

```
crl [tick]: {$\sigma, y_1, \ldots, y_n$} => {$\sigma$, max($y_1 - y, 0$), ..., max($y_n - y, 0$)} in time $y$
            if $y$ <= nextTimerExpires($y_1, \ldots, y_n$) and not transEnabled($\sigma, y_1, \ldots, y_n$) .
```

where σ, y_i, and y are all variables, `nextTimerExpires(`$y_1, \ldots, y_n$`)` returns the smallest non-zero value of the y_i, and `transEnabled(`$\sigma, y_1, \ldots, y_n$`)` holds iff some transition is enabled in the given state. The latter is defined by an equation

```
eq transEnabled($s, r_1, \ldots, r_n$) = true .
```

where r_i is 0 if $x_i \in X$ and is a variable y_i otherwise, for each transition $s \xrightarrow{a,X} s'$, and by having an equation that states that *otherwise* (i.e., if none of the above equations apply), `transEnabled(`$\sigma, y_1, \ldots, y_n$`)` is `false`:

```
eq transEnabled($\sigma, y_1, \ldots, y_n$) = false [owise] .
```

The SA in Fig. 1 (b) is therefore represented as a PRTRT as follows

```
rl [init]: init => {$s_0$, $y_1$, $y_2$} with probability $y_1$ := $F_1$ and $y_2$ := $F_2$ .
rl [push]: {$s_0$, 0, $y_2$} => {$s_1$, $y'_1$, $y_2$} with probability $y'_1$ := $F_1$ .
rl [pop] : {$s_1$, $y_1$, 0} => {$s_2$, $y_1$, $y'_2$} with probability $y'_2$ := $F_2$ .
rl [push]: {$s_1$, $y_1$, 0} => {$s_3$, $y_1$, $y'_2$} with probability $y'_2$ := $F_2$ .
rl [push]: {$s_2$, 0, $y_2$} => {$s_0$, $y'_1$, $y'_2$} with probability $y'_1$ := $F_1$ and $y'_2$ := $F_2$ .
rl [pop] : {$s_3$, 0, 0} => {$s_2$, 0, $y'_2$} with probability $y'_2$ := $F_2$ .

crl [tick]: {$\sigma$, $y_1$, $y_2$} => {$\sigma$, max($y_1$ - $y$, 0), max($y_2$ - $y$, 0)} in time $y$
            if $y$ <= nextTimerExpires($y_1$, $y_2$) and not transEnabled($\sigma$, $y_1$, $y_2$) .
```

where `transEnabled` and `nextTimerExpires` are defined by:

```
eq transEnabled($s_0$, 0, $y_2$) = true .   eq transEnabled($s_1$, $y_1$, 0) = true .
eq transEnabled($s_2$, 0, $y_2$) = true .   eq transEnabled($s_3$, 0, 0) = true .
eq transEnabled($s$, $y_1$, $y_2$) = false [owise] .
eq nextTimerExpires($y_1$, $y_2$) = if $y_1$ == 0 then (if $y_2$ == 0 then INF else $y_2$ fi)
                                else (if $y_2$ == 0 then $y_1$ else min($y_1$, $y_2$) fi) .
```

6.3 Deterministic and Stochastic Petri Nets

Deterministic and stochastic Petri nets (DSPNs) [2] are a fairly general class of timed Petri nets, where a transition can fire after having been continuously enabled for either a fixed (deterministic) or a random (exponentially distributed) amount of time. DSPNs are strictly more expressive than generalized stochastic Petri nets [3] (and, hence, stochastic Petri nets), which can be seen as DSPNs in which all deterministic transitions are instantaneous.

There are many variations of the basic model, including having inhibitor arcs, arc multiplicities that are functions of the marking, transition precedences, etc. To focus on the real-time and probabilistic aspects of the model, we assume a "standard" Petri net model extended with the above firing delays, and refer to [28] for the treatment of advanced Petri net features in rewriting logic. That is, a DSPN is a tuple (P, T, F, τ, R) where: P is a finite set of *places*; $T = T^D \uplus T^S$ is a finite set of *transitions*, partitioned into sets T^D and T^S of *deterministic* and *stochastic* transitions, respectively, and satisfying $T \cap P = \emptyset$; $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation; $\tau : T^D \to \mathbb{R}_{\geq 0}$ is a function that maps each *deterministic* transition t to its firing delay $\tau(t)$; if $\tau(t) = 0$ we call t an *instantaneous* transition; $R : T^S \to \mathbb{R}$ is a function that associates to each *stochastic* transition t the rate $R(t)$ of the exponential distribution of its firing delay. A transition must fire when it has been enabled continuously for the duration of its firing delay. We assume that the "enabled-time" of a transition is reset to zero when the transition is fired.

Figure 2 shows a DSPN specification, including its initial marking, of a client-server architecture. Stochastic (t_1), deterministic (t_3) and instantaneous (t_2) transitions are shown as empty rectangles, filled rectangles and thick lines, respectively. For the stochastic transitions we show the rate of the exponential distribution of their waiting times, while for the deterministic transitions we display the fixed amount of time associated with them.

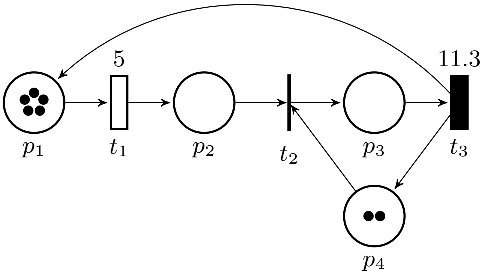


Fig. 2. A DSPN specifying a queueing model of a client-server architecture

Our representation follows the approach of [20], which sees a marking as a multiset of places and a transition as a multiset rewrite rule. In addition, for each transition $t \in T$, we associate a timer that denotes the remaining time during which the transition must be continuously enabled to fire. Such a timer can be represented by a term `<` t `;` r `>`, where t is the transition and r is its timer value. The global state of the system is therefore represented as a term `{`m`}`, where m is a multiset of places and transition timers, with multiset union denoted by juxtaposition. As a result of firing a transition, previously enabled transitions may be disabled, and vice versa. Therefore, we must recompute the transition timer values when a transition fires. For each transition t in the DSPN with pre-set $p_1\ p_2\ \ldots\ p_m$ and post-set $q_1\ q_2\ \ldots\ q_n$, we therefore have a rewrite rule

```
rl [apply-t] :
  {< t ; 0 >  p1  p2  ...  pm  REST}  =>  {recomputeTimers(< t ; INF >  q1  q2  ...  qn  REST)}.
```

which fires the transition t when its timer is 0. As a result, the pre-set is removed from the state, the post-set is added to it, t's timer is turned off (although it may be reset by `recomputeTimers` if the transition is still enabled, or re-enabled), and the function `recomputeTimers` is applied to the entire resulting state to recompute all transition timer values.

The function `recomputeTimers` is defined as follows: (i) if a transition is enabled and the corresponding timer is turned off (i.e., has the value `INF`), then the timer is reinitialized to the firing delay of the transition, otherwise the timer is left unchanged; and (ii) if a transition is not enabled, its timer is turned off. Case (i) can easily be defined by an equation defining `recomputeTimers` for *deterministic* transitions. However, an *equation* defining `recomputeTimers` cannot reset the timer of a stochastic transition, since the new timer value should be assigned probabilistically. Therefore, the timer of the stochastic transition is initialized to a new value `reset`, which will be replaced by a probabilistically chosen value in a *rewrite rule*. That is, for any *deterministic* transition t, case (i) above is defined by the following equation:

```
eq recomputeTimers(< t ; TI >  p1  p2  ...  pm  REST)
 = < t ; if TI == INF then τ(t) else TI fi >  recomputeTimers(p1  p2  ...  pm  REST) .
```

and for any *stochastic* transition t, case (i) is defined by the following equation:

```
eq recomputeTimers(< t ; TI >  p1  p2  ...  pm  REST)
 = < t ; if TI == INF then reset else TI fi >  recomputeTimers(p1  p2  ...  pm  REST) .
```

For each stochastic transition t with rate $R(t)$ we therefore have a rewrite rule

```
rl [set-stoc-timer]: < t ; reset >  =>  < t ; X > with probability X := ExpRate(R(t)) .
```

where the function `ExpRate(`λ`)` mimics the CDF of the exponential distribution with rate parameter $\lambda \in \mathbb{R}$. For case (ii), if the transition is not enabled, the following `owise` equation sets the corresponding timer to `INF`:

```
eq recomputeTimers(< T ; TI >  REST)  =  < T ; INF >  recomputeTimers(REST)  [owise] .
```

where `T` is a variable. Finally, we add the tick rule

```
crl [tick]: {SYSTEM}  =>  {decreaseTimers(SYSTEM, Y)} in time Y if Y <= mte(SYSTEM).
```

where decreaseTimers decreases the value of each timer by the elapsed time Y, and mte gives the smallest timer value (or 0 if a timer has the value reset). Therefore, this tick rule may advance time until the next timer expires. Appendix A gives a detailed specification of the PRTRT representation of a DSPN.

The representation of the DSPN in Fig. 2 contains the instantaneous rules

```
rl [apply-$t_1$] : {<$t_1$ ; 0> $p_1$ REST}  =>  {recomputeTimers(<$t_1$ ; INF> $p_2$ REST)} .
rl [apply-$t_2$] : {<$t_2$ ; 0> $p_2$ $p_4$ REST}  =>  {recomputeTimers(<$t_2$ ; INF> $p_3$ REST)} .
rl [apply-$t_3$] : {<$t_3$ ; 0> $p_3$ REST}  =>  {recomputeTimers(<$t_3$ ; INF> $p_1$ $p_4$ REST)} .
```

together with the equations defining the recomputeTimers function

```
eq recomputeTimers(<$t_1$ ; TI>  $p_1$ REST)
 =  <$t_1$ ; if TI == INF then reset else TI fi>  recomputeTimers($p_1$ REST) .
eq recomputeTimers(<$t_2$ ; TI>  $p_2$ $p_4$ REST)
 =  <$t_2$ ; if TI == INF then 0 else TI fi>  recomputeTimers($p_2$ $p_4$ REST) .
eq recomputeTimers(< $t_3$ ; TI >  $p_3$ REST)
 =  <$t_3$ ; if TI == INF then 11.3 else TI fi>  recomputeTimers($p_3$ REST) .
eq recomputeTimers(<$t$ ; TI>  REST)  =  <$t$ ; INF>  recomputeTimers(REST) [owise] .
ceq recomputeTimers(REST)  =  REST  if noTimers(REST) .
```

as well as the tick rule and the rule for setting the timer of the stochastic transition t_1 to a value sampled from the exponential distribution with rate 5:

```
rl [set-stoc-timer] : <$t_1$ ; reset>  =>  <$t_1$ ; X>  with probability X := ExpRate(5).
```

6.4 Handling Uncertainty in Probabilistic Transitions

We have identified two models for probabilistic real-time systems where the probability distribution associated with a transition is nondeterministically chosen from a *set* of probability distributions. We can represent these models as PRTRTs, which implies that PRTRTs are more expressive than (untimed) probabilistic rewrite theories in which the probability distribution is deterministically chosen. See [6] for details about the PRTRT representation of these models.

In *timed probabilistic transition systems* (TPTS) [29] the probability of making a transition belongs to an *interval*. This can be modeled in our formalism by exploiting the fact that, for a given rewrite rule r, π_r is a family of probability distributions, indexed both by the substitutions for the variables in the lefthand side of r and by the substitutions for the nondeterministically instantiated variables in the righthand side of r. In our PRTRT encoding we add the nondeterministically selected probability values to the state. Therefore, we specify the probabilistic transition from s_0 of the TPTS in Fig. 3 (a) by the following probabilistic tick rewrite rule, where σ, p and q are variables:

```
crl [tick] : {$s_0$; $q$} => {$\sigma$; $p$} in time 1 if $p \in [0.9, 1]$
                                    with probability $\sigma$ := $\begin{pmatrix} s_1 & s_2 \\ p & 1-p \end{pmatrix}$ .
```

In *timed probabilistic systems* (TPS) [10], the time that a node waits in a location—after selecting an outgoing action, but before *performing* the action—is a random value, whose *average* is given, but whose probability distribution is

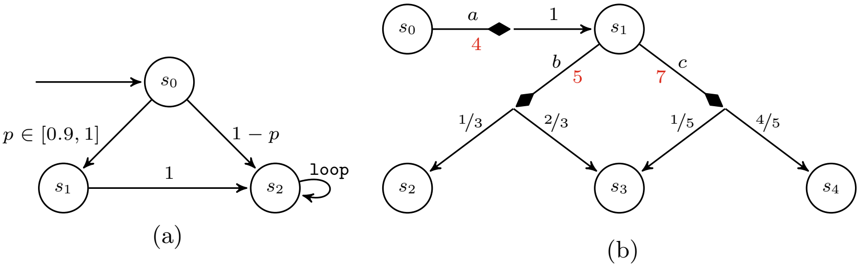


Fig. 3. (a) A timed probabilistic transition system (b) A timed probabilistic system

not specified. Figure 3 (b) shows an example of a TPS, where the actions are depicted as diamond-tipped arrows, the average time for performing an action is shown in red, and the probabilistic transitions are the arrows with probability values attached to them. We assume that the set of possible waiting times of action b is a finite[3] set $\{\varphi_1, \varphi_2, \ldots, \varphi_m\}$, where φ_i is selected with an *unknown* probability $p_i \in [0, 1]$. A state in the PRTRT representation of a TPS has the form $\texttt{\{}s, a, s', r, p_1; \ldots; p_n\texttt{\}}$, where s is the current state, a is the next action to perform, s' is the next state, r is the remaining time until the automaton performs the action a, and $p_1, \ldots, p_m$ are the nondeterministically chosen probability values as for TPTSs. The rule that performs action a when the timer expires and selects b as the next action is then

$$
\begin{aligned}
&\texttt{crl [}a_b\texttt{]: \{}s_0\texttt{,}\ a\texttt{,}\ s_1\texttt{, 0,}\ q_1\texttt{;}\ \ldots\texttt{;}\ q_n\texttt{\}} \quad \texttt{=>} \quad \texttt{\{}s_1\texttt{,}\ b\texttt{,}\ \sigma\texttt{,}\ r\texttt{,}\ p_1\texttt{;}\ \ldots\texttt{;}\ p_m\texttt{\}}\\
&\qquad \texttt{if}\ p_1 + \ldots + p_m = 1\ \texttt{and}\ p_1\varphi_1 + \ldots + p_m\varphi_m = 5\\
&\qquad \texttt{with probability}\ \sigma := \begin{pmatrix} s_2 & s_3 \\ 1/3 & 2/3 \end{pmatrix}\ \texttt{and}\ r \texttt{:=} \begin{pmatrix} \varphi_1 & \ldots & \varphi_m \\ p_1 & \ldots & p_m \end{pmatrix}\ .
\end{aligned}
$$

7 Concluding Remarks

We have defined the probabilistic real-time rewrite theory (PRTRT) formalism for modeling probabilistic real-time systems in rewriting logic, and have shown how PRTRTs can be seen as a unifying semantic framework in which a range of models for probabilistic real-time systems can be naturally represented, including systems with underspecified probability distributions. We have also given a PRTRT specification of a simple round trip time protocol that seems to be outside the class of systems that can be modeled using automaton-based formalisms, since the number of messages in a state can grow beyond any bound.

This work has provided the theoretical foundations for an analysis tool for probabilistic real-time systems in rewriting logic. In the future we must define property specification formalisms and implement suitable model checkers for PRTRTs. For this purpose, the statistical model checking approach seems very promising, since instead of performing exact probabilistic model checking—which often becomes unfeasible for large distributed systems—statistical model

[3] See [6] for the continuous case.

checking is typically much more efficient, although it only guarantees a property with a desired level of confidence. In particular, statistical model checking is based on evaluating a number of behaviors and is therefore easily parallelizable. Indeed, the PVeStA tool [4] provides a parallel statistical model checker for a subset of (untimed) probabilistic rewrite theories and could be a useful starting point for a future tool for PRTRTs.

References

1. Agha, G., Greenwald, M., Gunter, C.A., Khanna, S., Meseguer, J., Sen, K., Thati, P.: Formal modeling and analysis of DoS using probabilistic rewrite theories. In: Proc. FCS 2005 (2005)
2. Ajmone Marsan, M., Chiola, G.: On Petri nets with deterministic and exponentially distributed firing times. In: Rozenberg, G. (ed.) APN 1987. LNCS, vol. 266, Springer, Heidelberg (1987)
3. Ajmone Marsan, M., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Trans. Comput. Syst. 2 (1984)
4. AlTurki, M., Meseguer, J.: PVeStA: A parallel statistical model checking and quantitative analysis tool. To appear in Proc. CALCO 2011 (2011)
5. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126 (1994)
6. Bentea, L., Ölveczky, P.C.: Probabilistic real-time rewrite theories and their expressive power (2011), http://www.ifi.uio.no/~lucianb/publications/2011/prt-exp.pdf
7. Bruni, R., Meseguer, J.: Semantic foundations for generalized rewrite theories. Theoretical Computer Science 360(1-3) (2006)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
9. D'Argenio, P.R., Katoen, J.P., Brinksma, E.: An algebraic approach to the specification of stochastic systems. In: PROCOMET 1998. Chapman & Hall, Ltd., Boca Raton (1998)
10. De Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis, Stanford University, USA (1998)
11. Gilmore, S., Hillston, J.: The PEPA workbench: A tool to support a process algebra-based approach to performance modelling. In: Computer Performance Evaluation (1994)
12. Johnson, N.L., Kotz, S., Balakrishnan, N.: Continuous Univariate Distributions, 2nd edn. Wiley Series in Probability and Statistics, vol. 1. Wiley-Interscience, Hoboken (1994)
13. Katelman, M., Meseguer, J., Hou, J.: Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking. In: Barthe, G., de Boer, F.S. (eds.) FMOODS 2008. LNCS, vol. 5051, pp. 150–169. Springer, Heidelberg (2008)
14. Kumar, N., Sen, K., Meseguer, J., Agha, G.: Probabilistic rewrite theories: Unifying models, logics and tools. Technical report UIUCDCS-R-2003-2347, Department of Computer Science, University of Illinois at Urbana-Champaign (2003)
15. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic model checking for performance and reliability analysis. ACM SIGMETRICS Performance Evaluation Review 36(4) (2009)

16. Kwiatkowska, M., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theor. Comput. Sci. 282 (2002)
17. Lien, E., Ölveczky, P.C.: Formal modeling and analysis of an IETF multicast protocol. In: SEFM 2009. IEEE Computer Society, Los Alamitos (2009)
18. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science 96(1) (1992)
19. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (ed.) WADT 1997. LNCS, vol. 1376, Springer, Heidelberg (1998)
20. Meseguer, J., Montanari, U.: Petri nets are monoids. Information and Computation 88(2) (1990)
21. Ölveczky, P.C., Meseguer, J.: The Real-Time Maude tool. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 332–336. Springer, Heidelberg (2008)
22. Ölveczky, P.C., Meseguer, J.: Specification of real-time and hybrid systems in rewriting logic. Theoretical Computer Science 285(2), 359–405 (2002)
23. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20 (2007)
24. Ölveczky, P.C., Meseguer, J., Talcott, C.: Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. Formal Methods in System Design 29, 253–293 (2006)
25. Ölveczky, P.C., Thorvaldsen, S.: Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude. Theoretical Computer Science 410(2-3), 254–280 (2009)
26. Sen, K., Viswanathan, M., Agha, G.A.: VESTA: A statistical model-checker and analyzer for probabilistic systems. In: QEST 2005. IEEE Computer Society, Los Alamitos (2005)
27. Sproston, J.: Model Checking for Probabilistic Timed and Hybrid Systems. Ph.D. thesis, School of Computer Science, University of Birmingham (2001)
28. Stehr, M.O., Meseguer, J., Ölveczky, P.C.: Rewriting logic as a unifying framework for petri nets. In: Ehrig, H., Juhás, G., Padberg, J., Rozenberg, G. (eds.) APN 2001. LNCS, vol. 2128, pp. 250–303. Springer, Heidelberg (2001)
29. Yi, W.: Algebraic reasoning for real-time probabilistic processes with uncertain information. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) FTRTFT 1994 and ProCoS 1994. LNCS, vol. 863, pp. 680–693. Springer, Heidelberg (1994)
30. Zhang, H., Hou, J.: Maintaining Sensing Coverage and Connectivity in Large Sensor Networks. Ad Hoc & Sensor Wireless Networks 1(1-2) (2005)

A PRTRT Representation of DSPNs

We give below a more detailed specification, using Maude syntax, of the PRTRT representation of a DSPN.

```
sort Place .
ops p1 p2 p3 p4 ... : -> Place [ctor] .   --- one constant for each place

sort Transition .
ops t1 t2 t3 ... : -> Transition [ctor] . --- one constant for each transition
```

```
op reset : -> TimeInf [ctor] .              --- new 'timer' value

sort TransitionTimer .                      --- transition timers
op <_;_> : Transition TimeInf -> TransitionTimer [ctor] .

--- An extended marking is a multiset of places and transition timers:
sort ExtendedMarking .    subsort TransitionTimer Place < ExtendedMarking .
op none : -> ExtendedMarking [ctor] .       --- empty marking
--- assoc-comm multiset union operator:
op __ : ExtendedMarking ExtendedMarking -> ExtendedMarking
                                             [ctor assoc comm id: none] .

--- For EACH transition t we have a firing rule of the following form. Assume
--- that the preset of t is q1 ... qm and the postset of t is q1' ... qn', where
--- each qi and qk' is some place pj:

vars REST SYSTEM : ExtendedMarking .  var TI : TimeInf .  var T : Transition .
var X : Time .

rl [fire-t] :
   {< t ; 0 >  q1 ... qm REST}
   =>
   {recomputeTimers(< t ; INF >  q1' ... qn' REST)} .

op recomputeTimers : ExtendedMarking -> ExtendedMarking [frozen (1)] .

--- For EACH deterministic transition t, with the above pre- and postsets,
--- there is one equation as follows:

eq recomputeTimers(< t ; TI >  q1 ... qm REST)    --- t is enabled!
  =
   if TI == INF        --- t was previously disabled
     < t ; tau(t) >    --- initialize with value of firing delay
   else
     < t ; TI >        --- t was already enabled, do not change timer value
   fi
   recomputeTimers(q1 ... qm REST) .  --- recursively compute the other timers

--- For EACH stochastic transition t, with the above pre- and postsets,
--- there is one equation as follows, which is very similar to the
--- equation above, but resets the timer to 'reset':

eq recomputeTimers(< t ; TI >  q1 ... qm REST)    --- t is enabled
  =
   if TI == INF        --- t was previously disabled
     < t ; reset >     --- initialize with value 'reset'
   else
     < t ; TI >        --- t was already enabled, do not change timer value
   fi
   recomputeTimers(q1 ... qm REST) .  --- recursively compute the other timers

--- An owise equation matches when the transition T is not enabled.
--- Then we set the timer to 'INF', no mater its earlier value:
```

```
eq recomputeTimers(< T ; TI >  REST)  =  < T ; INF >  recomputeTimers(REST) .

--- Finally, when there are no transitions timers left to apply
--- recomputeTimers to, we are finished:

ceq recomputeTimers(REST)  =  REST  if noTimers(REST) .

--- This noTimers could also have been done with sorts, etc.
op noTimers : ExtendedMarking -> Bool .
eq noTimers(< T ; TI >  REST)  = false .
eq notimers(REST)  =  true [owise] .

--- Next, we instantiate the probabilistic variable; details
--- about the probability distribution omitted:

rl [instantiate-probabilistic-delay] :
   < T ; reset >  =>  < T ; X >  with probability X := distr(...R(T)...) .

--- Finally, the tick rule:
crl [tick] :
    {SYSTEM}  =>  {decreaseTimers(SYSTEM, X)}  in time X if X <= mte(SYSTEM) .

op decreaseTimers : ExtendedMarking Time -> ExtendedMarking [frozen (1)] .
eq decreaseTimers(< T ; TI >  REST, X)
  = < T ; TI - X >  decreaseTimers(REST, X) .
eq decreaseTimers(REST) = REST [owise] .    --- no more timers

--- The function mte gives the smallest timer value in the system:
op mte : ExtendedMarking -> TimeInf [frozen (1)] .
eq mte(< T ; TI >  REST) = min(if TI == reset then 0 else TI fi, mte(REST)) .
eq mte(REST) = INF [owise] .
```

Statistical Model Checking for Networks of Priced Timed Automata[⋆]

Alexandre David[1], Kim G. Larsen[1], Axel Legay[2], Marius Mikučionis[1],
Danny Bøgsted Poulsen[1], Jonas van Vliet[1], and Zheng Wang[3]

[1] Aalborg University, Denmark
[2] INRIA/IRISA Rennes, France
[3] East China Normal University, China

Abstract. This paper offers a natural stochastic semantics of Networks of Priced Timed Automata (NPTA) based on races between components. The semantics provides the basis for satisfaction of Probabilistic Weighted CTL properties (PWCTL), conservatively extending the classical satisfaction of timed automata with respect to TCTL. In particular the extension allows for hard real-time properties of timed automata expressible in TCTL to be refined by performance properties, e.g. in terms of probabilistic guarantees of time- and cost-bounded properties. A second contribution of the paper is the application of Statistical Model Checking (SMC) to efficiently estimate the correctness of non-nested PWCTL model checking problems with a desired level of confidence, based on a number of independent runs of the NPTA. In addition to applying classical SMC algorithms, we also offer an extension that allows to efficiently compare performance properties of NPTAs in a parametric setting. The third contribution is an efficient tool implementation of our result and applications to several case studies.

1 Introduction

Model Checking (MC) [11] is a widely recognised approach to guarantee the correctness of a system by checking that any of its behaviors is a model for a given property. There are several variants and extensions of MC aiming at handling real-time and hybrid systems with quantitative constraints on time, energy or more general continuous aspects [1, 2, 3, 6]. Within the field of embedded systems these formalisms and their supporting tools [16, 29, 30, 32] are now successfully applied to time- and energy-optimal scheduling, WCET analysis and schedulability analysis.

Compared with traditional approaches, a strong point of real-time model checking is that it (in principle) only requires a model to be applicable, thus extensions to multi-processor setting is easy. A weak point of model checking is the state-space explosion, i.e. the exponential growth in the analysis effort measured in the number of model-components. Another limitation of real-time

[⋆] Work partially supported by VKR Centre of Excellence – MT-LAB and by an "Action de Recherche Collaborative" ARC (TP)I.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 80–96, 2011.

model checking is that it merely provides – admittedly most important – hard quantitative guarantees, e.g. the worst case response time of a recurrent task under a certain scheduling principle, the worst case execution time of a piece of code running on a particular execution platform, or the worst case time before consensus is reached by a real-time network protocol. In addition to these hard guarantees, it would be desirable in several situations to obtain refined performance information concerning likely or expected behaviors in terms of timing and resource consumption. In particular, this would allow to distinguish and select between systems that perform identically from a worst-case perspective.

As a first contribution we propose a stochastic semantics for Priced Timed Automata (PTA), whose clocks can evolve with different rates, while[1] being used with no restrictions in guards and invariants. Networks of PTAs (NPTA) are created by composing PTAs via input and output actions. More precisely, we define a natural stochastic semantics for networks of NPTAs based on races between components being composed. We shall observe that such race can generate arbitrarily complex stochastic behaviors from simple assumptions on individual components. We shall see that our semantics cannot be emulated by applying the existing stochastic semantic of [4, 8] to the product of components. Other related work includes the very rich framework of stochastic timed systems of MoDeST [10]. Here, however, general hybrid variables are not considered and parallel composition does not yield fully stochastic models. For the notion of probabilistic hybrid systems considered in [31] the choice of time is resolved non-deterministically rather than stochastically as in our case. Moreover, based on the stochastic semantics, we are able to express refined performance properties, e.g. in terms of probabilistic guarantees of time- and cost-bounded properties[2].

To allow for the efficient analysis of probabilistic performance properties we propose to work with Statistical Model Checking (SMC) [28, 35], an approach that has been proposed as an alternative to avoid an exhaustive exploration of the state-space of the model. The core idea of SMC is to monitor some simulations of the system, and then use results from the statistic area (including sequential hypothesis testing or Monte Carlo simulation) in order to decide whether the system satisfies the property with some degree of confidence.

Thus, as a second contribution, we provide an efficient implementation of several existing SMC algorithms that we use for checking the correctness of NPTAs with respect to a stochastic extension of cost-constrained temporal logic – this extension being conservative with respect to the classical (non-stochastic) interpretation of the logic. We shall observe that two timed bisimilar NPTAs may be distinguisable by PWCTL. The series of algorithms we implemented includes the sequential hypothesis test by Wald [34] as well as a quantitative approach [18]. Our implementation relies on a new efficient algorithm for generating runs of NPTAs in a random manner. In addition, we also propose another SMC algorithm to compare the probabilities of two properties without computing them individually – which is useful to compare the performances of a program with

[1] In contrast to the usual restriction of priced timed automata [3, 6].

[2] Clocks with different rates can be used to model costs.

one of its evolutions at cheap cost. This probability comparison problem, which is far beyond the scope of existing time model checking approaches, can be approximated with an extension of the sequential hypothesis testing and has the advantage of unifying the confidence in the comparison. In addition to be the first to apply such extension in the context of formal verification, we also propose a new variant that allows to reuse existing results in parallel when comparing the properties on different timed bounds.

Finally, one of the most interesting contribution of our work takes the form of a series of new case studies that are analyzed with a new stochastic extension of Uppaal [13]. Particularly, we show how our approach can be used to resolve scheduling problems. Such problems are defined using Duration Probabilistic Automata (DPA) [24], a new and natural model for specifying list of tasks and shared resources. We observe that our approach is not only more general, but also an order of magnitude faster than the hypothesis testing engine recently implemented in the Prism toolset. Our work thus presents significant advances in both the modeling and the efficient verification of network of complex systems.

Related Work. Some works on probabilistic semantics of timed automata have already been discussed above. Simulation-based approaches such as Monte Carlo have been in use since decades, however the use of simulation and hypothesis testing to reason on formal models is a more recent advance. First attempts to apply hypothesis testing on stochastic extension of Hennessy-Milner logic can be found in [23]. In [35, 37], Younes was the first to apply hypothesis testing to stochastic systems whose properties are specified with (bounded) temporal logic. His approach is implemented in the Ymer toolset [36] and can be applied on time-homogeneous generalized semi-Markov processes, while our semantics addresses the composition of stochastic systems allowing to compose a global system from components and reason about communication between independent processes. In addition to Younes work we explore continuous-time features, formalize and implement Wald's ideas where the probability comparison can be evaluated on NPTA processes. In a recent work [38], Zuliani et al. extended the SMC approach to hybrid systems. Their work is a combination of [20] and [12] based on Simulink models (non-linear hybrid systems), whereas our method is specialised to networks of priced timed automata where model-checking techniques can be directly applicable using the same tool suite. In addition we provide means of comparing performances without considering individual probabilities. Finally, a very recent work [9] proposes partial order reduction techniques to resolve non-determinism between components rather than defining a unique stochastic distribution on their product behaviors. While this work is of clear interest, we point out that the application of partial order may considerably increase the computation time and for some models partial orders cannot resolve non-determinism, especially when considering continuous time [25]. Finally, we mention [22] that proposes a stochastic semantics to Uppaal's models through simulation. This work does not consider race between components and offers no tool implementation.

2 Network of Priced Timed Automata

We consider the notion of *Networks of Priced Timed Automata (NPTA)*, generalizing that of regular timed automata (TA) in that clocks may have different rates in different locations. In fact, the expressive power (up to timed bisimilarity) of NPTA equals that of general linear hybrid automata (LHA) [1], rendering most problems – including that of reachability – undecidable.

Let X be a finite set of variables, called *clocks*[3]. A *clock valuation* over X is a mapping $\nu : X \rightarrow \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ is the set of nonnegative reals. We write $\mathbb{R}_{\geq 0}^X$ for the set of clock valuations over X. Let $r : X \rightarrow \mathbb{N}$ be a *rate vector*, assigning to each clock of X a rate. Then, for $\nu \in \mathbb{R}_{\geq 0}^X$ and $d \in \mathbb{R}_{\geq 0}$ a delay, we write $\nu + r \cdot d$ for the clock valuation defined by $(\nu + r \cdot d)(x) = \nu(x) + r(x) \cdot d$ for any clock $x \in X$. We denote by $\mathbb{N}^X$ the set of all rate vectors. If $Y \subseteq X$, the valuation $\nu[Y]$ is the valuation assigning 0 when $x \in Y$ and $\nu(x)$ when $x \notin Y$. An *upper bounded (lower bound) guard* over X is a finite conjunction of simple clock bounds of the form $x \sim n$ where $x \in X$, $n \in \mathbb{N}$, and $\sim \in \{<, \leq\}$ ($\sim \in \{>, \geq\}$) We denote by $\mathcal{U}(X)$ ($\mathcal{L}(X)$ the set of upper (lower) bound guards over X, and write $\nu \models g$ whenever ν is a clock valuation satisfying the guard g. Let $\Sigma = \Sigma_i \uplus \Sigma_o$ be a disjoint sets of input and output actions.

Definition 1. *A* Priced Timed Automaton *(PTA) is a tuple* $\mathcal{A} = (L, \ell_0, X, \Sigma, E, R, I)$ *where: (i)* L *is a finite set of locations, (ii)* $\ell_0 \in L$ *is the initial location, (iii)* X *is a finite set of clocks, (iv)* $\Sigma = \Sigma_i \uplus \Sigma_o$ *is a finite set of actions partitioned into inputs (*Σ_i*) and outputs (*Σ_o*), (v)* $E \subseteq L \times \mathcal{L}(X) \times \Sigma \times 2^X \times L$ *is a finite set of edges, (vi)* $R : L \rightarrow \mathbb{N}^X$ *assigns a rate vector to each location, and (viii)* $I : L \rightarrow \mathcal{U}(X)$ *assigns an invariant to each location.*

The semantics of NPTAs is a timed labelled transition system whose states are pairs $(\ell, \nu) \in L \times \mathbb{R}_{\geq 0}^X$ with $\nu \models I(\ell)$, and whose transitions are either delay $(\ell, \nu) \xrightarrow{d} (\ell, \nu')$ with $d \in \mathbb{R}_{\geq 0}$ and $\nu' = \nu + R(\ell) \cdot d$, or discrete $(\ell, \nu) \xrightarrow{a} (\ell', \nu')$ if there is an edge (ℓ, g, a, Y, ℓ') such that $\nu \models g$ and $\nu' = \nu[Y]$. We write $(\ell, \nu) \rightsquigarrow (\ell', \nu')$ if there is a finite sequence of delay and discrete transitions from (ℓ, ν) to (ℓ', ν').

Networks of Priced Timed Automata. Following the compositional specification theory for timed systems in [14], we shall assume that NPTAs are: (1)[*Input-enabled*:] for all states (ℓ, ν) and input actions $\iota \in \Sigma_i$, for all TAs j, there is an edge $(\ell^j, g, \iota, Y, {\ell^j}')$ such that $\nu \models g$, (2) [*Deterministic*:] for all states (ℓ, ν) and actions $a \in \Sigma$, whenever $(\ell, \nu) \xrightarrow{a} (\ell', \nu')$ and $(\ell, \nu) \xrightarrow{a} (\ell'', \nu'')$ then $\ell' = \ell''$ and $\nu' = \nu''$, and (3) [*Non-zenos*:] time always diverge. Moreover, different automata synchronize on matching inputs and outputs as a standard broadcast synchronization[17].

Whenever $\mathcal{A}^j = (L^j, X^j, \Sigma^j, E^j, R^j, I^j)$ $(j = 1 \ldots n)$ are NPTA, they are *composable* into a *closed network* iff their clock sets are disjoint ($X^j \cap X^k = \emptyset$

[3] We will (mis)use the term "clock" from timed automata, though in the setting of NPTAs the variables in X are really general real-valued variables.

when $j \neq k$), they have the same action set ($\Sigma = \Sigma^j = \Sigma^k$ for all j, k), and their output action-sets provide a partition of Σ ($\Sigma_o^j \cap \Sigma_o^k = \emptyset$ for $j \neq k$, and $\Sigma = \cup_j \Sigma_o^j$). For $a \in \Sigma$ we denote by $c(a)$ the unique j with $a \in \Sigma^j$.

Definition 2. *Let $\mathcal{A}^j = (L^j, X^j, \Sigma, E^j, R^j, I^j)$ (with $j = 1 \ldots n$) be composable NPTAs. Their* composition $(\mathcal{A}_1 \,|\, \ldots \,|\, \mathcal{A}_n)$ *is the NPTA $\boldsymbol{\mathcal{A}} = (L, X, \Sigma, E, R, L)$ where (i) $L = \times_j L^j$, (ii) $X = \cup_j X^j$, (iii) $R(\boldsymbol{\ell})(x) = R^j(\ell^j)(x)$ when $x \in X^j$, (iv) $I(\boldsymbol{\ell}) = \cap_j I(\ell^j)$, and (v) $(\boldsymbol{\ell}, \cap_j g_j, a, \cup_j r_j, \boldsymbol{\ell'}) \in E$ whenever $(\ell_j, g_j, a, r_j, \ell'_j) \in E^j$ for $j = 1 \ldots n$.*

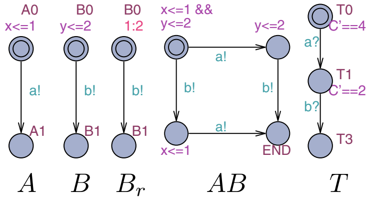


Fig. 1. Three composable NPTAs: A, B and T; A, B_r and T; and AB and T

Example 1. Let A, B, T and AB be the priced timed automata depicted in Fig. 1[4] Then A, B and T are composable as well as AB and T. In fact the composite systems $(A|B|T)$ and $(AB|T)$ are timed (and priced) bisimilar, both having the transition sequence:

$$\big((A_0, B_o, T_0), [x = 0, y = 0, C = 0]\big) \xrightarrow{1}\xrightarrow{a!}$$
$$\big((A_1, B_0, T_1), [x = 1, y = 1, C = 4]\big) \xrightarrow{1}\xrightarrow{b!}$$
$$\big((A_1, B_1, T_2), [x = 2, y = 2, C = 6]\big),$$

demonstrating that the final location T_3 of T is reachable with cost 6.

3 Probabilistic Semantics of NPTA

Continuing Example 1 we may realise that location T_3 of the component T is reachable within cost 0 to 6 and within total time 0 and 2 in both $(A|B|T)$ and $(AB|T)$ depending on when (and in which order) A and B (AB) chooses to perform the output actions $a!$ and $b!$. Assuming that the choice of these time-delays is governed by probability distributions, we will in this section define a probability measure over sets of infinite runs of networks of NPTAs.

In contrast to the probabilistic semantics of timed automata in [4, 8] our semantics deals with networks and thus with races between components. Let $\mathcal{A}^j = (L^j, X^j, \Sigma, E^j, R^j, I^j)$ $(j = 1 \ldots n)$ be a collection of composable NPTAs. Under the assumption of input-enabledness, disjointness of clock sets and output actions, states of the the composite NPTA $\boldsymbol{\mathcal{A}} = (\mathcal{A}_1 \,|\, \ldots \,|\, \mathcal{A}_n)$ may be seen as tuples $\mathbf{s} = (s_1, \ldots, s_n)$ where s_j is a state of $\mathcal{A}^j$, i.e. of the form (ℓ, ν) where $\ell \in L^j$ and $\nu \in \mathbb{R}_{\geq 0}^{X^j}$. Our probabilistic semantics is based on the principle of independency between components. Repeatedly each component decides on its own – based on a given delay density function and output probability function – how much to delay before outputting and what output to broadcast at that moment. Obviously, in such a race between components the outcome will be determined by the component that has chosen to output after the minimum delay: the output is broadcast and all other components may consequently change state.

[4] The broadcast synchronization we use allows us to ignore missing input transitions that may otherwise be added as looping transitions.

Probabilistic Semantics of NPTA Components. Let us first consider a component $\mathcal{A}^j$ and let St^j denote the corresponding set of states. For each state $s = (\ell, \nu)$ of $\mathcal{A}^j$ we shall provide probability distributions for both delays and outputs. In this presentation, we restrict to uniform and universal distributions, but arbitrary distributions can be considered.

The *delay density function* μ_s over delays in $\mathbb{R}_{\geq 0}$ will be either a uniform or an exponential distribution depending on the invariant of ℓ. Denote by E_ℓ the disjunction of guards g such that $(\ell, g, o, -, -) \in E^j$ for some output o. Denote by $d(\ell, \nu)$ the infimum delay before enabling an output, i.e. $d(\ell, \nu) = \inf\{d \in \mathbb{R}_{\geq 0} : \nu + R^j \cdot d \models E_\ell\}$, and denote by $D(\ell, \nu)$ the supremum delay, i.e. $D(\ell, \nu) = \sup\{d \in \mathbb{R}_{\geq 0} : \nu + R^j \cdot d \models I^j(\ell)\}$. If $D(\ell, \nu) < \infty$ then the delay density function μ_s is a uniform distribution on $[d(\ell, \nu), D(\ell, \nu)]$. Otherwise – that is $I^j(\ell)$ does not put an upper bound on the possible delays out of s – the delay density function μ_s is an exponential distribution with a rate $P(\ell)$, where $P : L^j \to \mathbb{R}_{\geq 0}$ is an *additional* distribution rate component added to the NPTA $\mathcal{A}^j$. For every state $s = (\ell, \nu)$, the *output probability function* γ_s over Σ_o^j is the uniform distribution over the set $\{o : (\ell, g, o, -, -) \in E^j \wedge \nu \models g\}$ whenever this set is non-empty[5]. We denote by s^o the state after the output of o. Similarly, for every state s and any input action ι, we denote by s^ι the state after having received the input ι.

Probabilistic Semantics of Networks of NPTA. We shall now see that while the stochastic semantics of each PTA is rather simple (but quite realistic), arbitrarily complex stochastic behavior can be obtained by their composition.

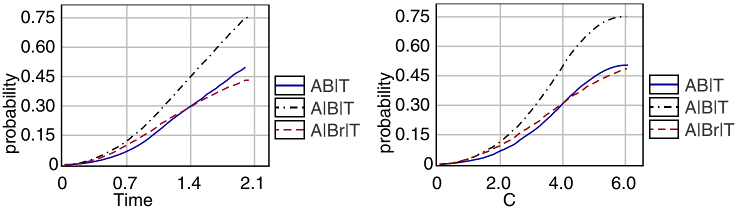
Reconsider the closed network $\boldsymbol{\mathcal{A}} = (\mathcal{A}_1 \,|\, \dots \,|\, \mathcal{A}_n)$ with a state space $\mathsf{St} = \mathsf{St}_1 \times \cdots \times \mathsf{St}_n$. For $\mathbf{s} = (s_1, \dots, s_n) \in \mathsf{St}$ and $a_1 a_2 \dots a_k \in \Sigma^*$ we denote by $\pi(\mathbf{s}, a_1 a_2 \dots a_k)$ the set of all maximal runs from $\mathbf{s}$ with a prefix $t_1 a_1 t_2 a_2 \dots t_k a_k$ for some $t_1, \dots, t_n \in \mathbb{R}_{\geq 0}$, that is runs where the i'th action a_i has been outputted by the component $A_{c(a_i)}$. We now inductively define the following measure for such sets of runs:

$$\mathbb{P}_{\boldsymbol{\mathcal{A}}}\big(\pi(\mathbf{s}, a_1 \dots a_n)\big) = \int_{t \geq 0} \mu_{s_c}(t) \cdot \big(\prod_{j \neq c} \int_{\tau > t} \mu_{s_j}(\tau) d\tau \big) \cdot \gamma_{s_c{}^t}(a_1) \cdot \mathbb{P}_{\boldsymbol{\mathcal{A}}}\big(\pi(\mathbf{s}^t)^{a_1}, a_2 \dots a_n)\big) dt$$

where $c = c(a_1)$, and as base case we take $P_{\boldsymbol{\mathcal{A}}}(\pi(\mathbf{s}), \varepsilon) = 1$.

This definition requires a few words of explanation: at the outermost level we integrate over all possible initial delays t. For a given delay t, the outputting component $c = c(a_1)$ will choose to make the broadcast at time t with the stated density. Independently, the other components will choose to a delay amount, which – in order for c to be the winner – must be larger than t; hence the product of the probabilities that they each make such a choice. Having decided for making the broadcast at time t, the probability of actually outputting a_1 is included. Finally, in the global state resulting from all components having delayed t time-units and changed state according to the broadcasted action a_1 the probability of runs according to the remaining actions $a_2 \dots a_n$ is taken into account.

[5] Otherwise a specific weight distribution can be specified and used instead.



(a) (b)

Fig. 2. Cumulative probabilities for time and cost-bounded reachability of T_3

Logical Properties. Following [26], the measure $\mathbb{P}_{\boldsymbol{\mathcal{A}}}$ may be extended in a standard and unique way to the σ-algebra generated by the sets of runs (so-called cylinders) $\pi(\mathbf{s}, a_1 a_2 \dots a_n)$. As we shall see this will allow us to give proper semantics to a range of probabilistic time- and cost-constrained temporal properties. Let $\mathcal{A}$ be a NPTA. Then we consider the following non-nested PWCTL properties:

$$\psi \ ::= \ \mathbb{P}\big(\Diamond_{C \leq c}\varphi\big) \sim p \ \mid \ \mathbb{P}\big(\Box_{C \leq c}\varphi\big) \sim p$$

where C is an observer clock (of $\mathcal{A}$), φ a state-property (wrt. $\mathcal{A}$) , $\sim \in \{<, \leq, =, \geq, >\}$, and $p \in [0,1]$. This logic is a stochastic extension of the classical WCTL logic for non-stochastic systems, where the existential quantifier is replaced by a probability operator. For the semantics let $\mathcal{A}^*$ be the modification of $\mathcal{A}$, where the guard $C \leq c$ has been conjoined to the invariant of all locations and an edge $(\ell, \varphi, o_\varphi, \emptyset, \ell)$ has been added to all locations ℓ, where o_φ is a new output action. Then:

$$\mathcal{A} \models \mathbb{P}\big(\Diamond_{C \leq c}\varphi\big) \sim p \ \text{ iff } \ \mathbb{P}_{\mathcal{A}^*}\Big(\bigcup_{\sigma \in \Sigma^*} \pi(s_0, \sigma o_\varphi)\Big) \sim p$$

which is well-defined since the σ-algebra on which $\mathbb{P}_{\mathcal{A}^*}$ is defined is closed under countable unions and finite intersections. To complete the semantics, we note that $\mathbb{P}(\Box_{C \leq c}\varphi) \sim p$ is equivalent to $(1-p) \sim \mathbb{P}(\Diamond_{C \leq c}\neg\varphi)$.[6]

Compared with previous stochastic semantics of timed automata (see e.g., [4, 8]), we emphasize the novelty of the semantics of NPTA in terms of RACES between components, truthfully reflecting their independencies. In particular our stochastic semantics of a network $(A_1|..|A_n)$ is significantly different from that obtained by applying the stochastic semantics of [4, 8] to a product construction $A_1 A_2 \dots A_n$, as information about independencies are lost. So though $(A_1|..|A_n)$ and $A_1 A_2 \dots A_n$ are timed bisimilar they are in general not probabistic timed bisimilar, and hence distinguishable by PWCTL. The situation is illustrated with the following example.

Example 2. Reconsider the Example of Fig. 1. Then it can be shown that $(A|B|T) \models \mathbb{P}\big(\Diamond_{t \leq 2} T_3\big) = 0.75$ and $(A|B|T) \models \mathbb{P}\big(\Diamond_{C \leq 6} T_3\big) = 0.75$, whereas

[6] We also note that the above (stochastic) interpretation of PWCTL is a conservative extension of the classical (non-stochastic) interpretation of WCTL, in the sense that $\mathcal{A} \models \mathbb{P}\big(\Diamond_{C \leq c}\varphi\big) > 0$ implies $\mathcal{A}_n \models \mathsf{E}\Diamond_{C \leq c}\varphi$, where $\mathcal{A}_n$ refers to the standard non-stochastic semantics of $\mathcal{A}$.

$(AB|T) \models \mathbb{P}\big(\Diamond_{t\leq 2}T_3\big) = 0.50$ and $(AB|T) \models \mathbb{P}\big(\Diamond_{C\leq 6}T_3\big) = 0.50$. Fig. 2 gives a time- and cost-bounded reachability probabilities for $(A|B|T)$ and $(AB|T)$ for a range of bounds. Thus, though the two NPTAs satisfy the same WCTL properties, they are obviously quite different with respect to PWCTL. The NPTA B_r of Fig. 1 is a variant of B, with the uniform delay distribution enforced by the invariant $y \leq 2$ being replaced by an exponential distribution with rate $\frac{1}{2}$. Here $(A|B_r|T)$ satisfies $\mathbb{P}\big(\Diamond_{t\leq 2}T_3\big) \approx 0.41$ and $\mathbb{P}\big(\Diamond_{C\leq 6}T_3\big) \approx 0.49$.

4 Statistical Model Checking for NPTA

As we pointed out, most of model checking problems for NPTAs and PWCTL (including reachability) are undecidable. Our solution is to use a technique that approximates the answer. We rely on *Statistical Model Checking* (SMC)[28, 35], that is a series of simulation-based techniques that generate runs of the systems, monitor them, and then use algorithms from statistics to get an estimate of the entire system. At the heart of any SMC approach, there is an algorithm used to generate runs of the system following a stochastic semantics. We propose such an algorithm for NPTAs corresponding to the stochastic semantics proposed in Section 3. Then, we recap existing statistic algorithms, providing the basis for a first SMC algorithm for NPTAs.

Generating Runs of NPTA. SMC is used for properties that can be monitored on finite runs. Here, we propose an algorithm that given an NPTA generates a random run up to a cost bound c (with time bounds being a simple case) of an observer clock C. A run of a NPTA is a sequence of alternations of states $\boldsymbol{s_0} \xrightarrow{d_0} \boldsymbol{s_0'} \xrightarrow{o_0} \boldsymbol{s_1} \xrightarrow{d_1} \ldots \boldsymbol{s}_n$ obtained by performing delays d_i and emitting outputs o_i. Here we consider a network of NPTAs with states being of the form $(\boldsymbol{\ell}, \nu)$. We construct random runs according to Algorithm 1. We start from an initial state $(\boldsymbol{\ell_0}, \nu_0)$ and repeatedly concatenate random successor states until we reach the bound c for the given observer clock C. Recall that $\nu(C)$ is the value of C in state $(\boldsymbol{\ell}, \nu)$, and the rate of C in location $\boldsymbol{\ell}$ is $R(C)(\boldsymbol{\ell})$. We use the notation $\oplus$ to concatenate runs and $tail(run)$ to access the last state of a run and $delay(\mu_s)$ returns a random delay according to the delay density function μ_s as described in Section 3. The statement "pick" means choose uniformly among the possible choices. Lines 5-6 stop the delay when the runs reach their time bounds with the values of the clocks depending on their rates. The Algorithm 1 may be seen to be correct with respect to the stochastic semantics of NPTAs given in Section 3 in the sense that the probability of the (random) run $RR_{\boldsymbol{\mathcal{A}}}\big((\boldsymbol{\ell_0}, \nu_0), C, c\big)$ satisfying $\Diamond_{C\leq c}\varphi$ is $\mathbb{P}_{\boldsymbol{\mathcal{A}}}\Big(\Diamond_{C\leq c}\,\varphi\Big)$.

Statistical Model Checking Algorithms. We briefly recap statistical algorithms permitting to answer the following two types of questions : (1) *Qualitative :* Is the probability for a given NPTA $\boldsymbol{\mathcal{A}}$ to satisfy a property $\Diamond_{C\leq c}\varphi$ greater or equal to a certain threshold θ ? and (2) *Quantitative :* What is the probability for $\boldsymbol{\mathcal{A}}$ to satisfy $\Diamond_{C\leq c}\varphi$. Each run of the system is encoded as a Bernoulli random variable that is true if the run satisfies the property and false otherwise.

Algorithm 1. Random run for a NPTA-network $\boldsymbol{\mathcal{A}}$

```
function $RR_{\boldsymbol{\mathcal{A}}}((\boldsymbol{\ell_0}, \nu_0), C, c)$
$run := (\boldsymbol{\ell}, \nu) := tail(run) := (\boldsymbol{\ell_0}, \nu_0)$
while $\nu(C) < c$ do
    for $i = 1$ to $|\boldsymbol{\ell}|$ do $d_i := delay(\mu_{(\ell_i,\nu_i)})$
    $d := min_{1 \leq i \leq |\boldsymbol{\ell}|}(d_i)$
    if $d = +\infty \vee \nu(C) + d * R(\boldsymbol{\ell})(C) \geq c$ then
        $d := (\nu(C) - c)/R(\boldsymbol{\ell})(C)$
        return $run \oplus \xrightarrow{d} (\boldsymbol{\ell}, \nu + d * R(\boldsymbol{\ell}))$
    end
    else
        pick $k$ such that $d_k = d$; $\nu_d := \nu + d * R(\boldsymbol{\ell})$
        pick $\ell_k \xrightarrow{g,o,r} \ell'_k$ with $g(\nu_d)$
        $run := run \oplus \xrightarrow{d} (\boldsymbol{\ell}, \nu_d) \xrightarrow{g,o,r} (\boldsymbol{\ell}[l'_k/l_k], [r \mapsto 0](\nu_d))$
    end
    $(\boldsymbol{\ell}, \nu) := tail(run)$
end
return $run$
```

Qualitative Question. This problem reduces to test the hypothesis $H : p = \mathbb{P}_{\boldsymbol{\mathcal{A}}}(\Diamond_{C \leq c}\varphi) \geq \theta$ against $K : p < \theta$. To bound the probability of making errors, we use strength parameters α and β and we test the hypothesis $H_0 : p \geq p_0$ and $H_1 : p \leq p_1$ with $p_0 = \theta + \delta_0$ and $p_1 = \theta - \delta_1$. The interval $p_0 - p_1$ defines an indifference region, and p_0 and p_1 are used as thresholds in the algorithm. The parameter α is the probability of accepting H_0 when H_1 holds (false positives) and the parameter β is the probability of accepting H_1 when H_0 holds (false negatives). The above test can be solved by using Wald's sequential hypothesis testing [34]. This testcomputes a proportion r among those runs that satisfy the property. With probability 1, the value of the proportion will eventually cross $\log(\beta/(1-\alpha)$ or $\log((1-\beta)/\alpha)$ and one of the two hypothesis will be selected.

Quantitative Question. This algorithm [19] computes the number N of runs needed in order to produce an approximation interval $[p-\epsilon, p+\epsilon]$ for $p = Pr(\psi)$ with a confidence $1-\alpha$. The values of ϵ and α are chosen by the user and N relies on the Chernoff-Hoeffding bound.

5 Beyond "Classical" Statistical Model-Checking

Here, we want to compare $p_1 = \mathbb{P}_{\boldsymbol{\mathcal{A}}}(\Diamond_{C_1 \leq c_1}\varphi_1)$ and $p_2 = \mathbb{P}_{\boldsymbol{\mathcal{A}}}(\Diamond_{C_2 \leq c_2}\varphi_2)$ without computing them. This comparison has clear practical applications e.g. it can be used to compare the performances of an original program with one of its newly designed extensions. This comparison cannot be performed with the algorithm presented in the previous section. Moreover, using Monte Carlo to estimate the probabilities (which is costly) would not help as both such probabilities would be estimated with different confidences that could hardly be related[7]. In [34], Wald has shown that this problem can be reduced to a sequential hypothesis testing one. Our contributions here are (1) to apply this algorithm in the formal verification area, (2) to extend the original algorithm of [34] to handle cases where we observe the same outcomes for both experiments, and (3) to implement a parametric extension of the algorithm that allows to reuse results on several timed bounds. More precisely, instead of comparing two probabilities with one common cost bound $C \leq c$, the new extension does it for all the N bounds $i * c/N$ with $i = 1 \dots N$ by reusing existing runs.

Comparison Algorithm. Let the efficiency of satisfying $\Diamond_{C_1 \leq c_1} \varphi_1$ over runs be given by $k_1 = p_1/(1-p_1)$ and similarly for $\Diamond_{C_2 \leq c_2} \varphi_2$. The relative superiority of "φ_2 over φ_1" is measured by the ratio $u = \frac{k_2}{k_1} = \frac{p_2(1-p_1)}{p_1(1-p_2)}$. If $u = 1$ both properties are equally good, if $u > 1$, φ_2 is better, otherwise φ_1 is better. Due to indifference region, we have two parameters u_0 and u_1 such that $u_0 < u_1$ to make the decision. If $u \leq u_0$ we favor φ_1 and if $u \geq u_1$ we favor φ_2. The parameter α is the probability of rejecting φ_1 when $u \leq u_0$ and the parameter β is the probability of rejecting φ_2 when $u \geq u_1$. An outcome for the comparison algorithm is a pair $(x_1, x_2) = (r_1 \models \Diamond_{C_1 \leq c_1} \varphi_1, r_2 \models \Diamond_{C_2 \leq c_2} \varphi_2)$ for two independent runs r_1 and r_2. In Wald's version (lines 10–14 of Algorithm 2), the outcomes $(0, 0)$ and $(1, 1)$ are ignored. The algorithm works if it is guaranteed to eventually generate different outcomes. We extend the algorithm with a qualitative test (lines 5–9 of Algorithm 2) to handle the case when the outcomes are always the same. The hypothesis we test is $\mathbb{P}_{\boldsymbol{\mathcal{A}}}((r_1 \models \Diamond_{C_1 \leq c_1} \varphi_1) = (r_2 \models \Diamond_{C_2 \leq c_2} \varphi_2)) \geq \theta$ for two independent runs r_1 and r_2. We note that this does not affect the correctness of the original algorithm for accepting or rejecting process 2. The modified algorithm now returns *indifferent* in addition, which corresponds to our added hypothesis to cut down the number of necessary runs[8]. Typically we want the parameters $p_0' = \theta + \delta_0$ (for the corresponding hypothesis H_0) and $p_1' = \theta - \delta_1$ (for H_1) to be close to 1. Our version of the comparison algorithm is shown in algorithm 2 with the following initializations:

$$a = \frac{\log(\frac{\beta}{1-\alpha})}{\log(u_1) - \log(u_0)}, r = \frac{\log(\frac{1-\beta}{\alpha})}{\log(u_1) - \log(u_o)}, c = \frac{\log(\frac{1+u_1}{1+u_0})}{\log(u_1) - \log(u_o)}$$

Parametrised Comparisons. We now generalise the comparison algorithm to give answers not only for one cost bound c but N cost bounds $i * c/N$ (with $i = 1 \dots N$). This algorithm is of particular interest to generate distribution over timed bounds value of the property. The idea is to reuse the runs of smaller bounds. When $\Diamond_{C \leq c} \varphi_1$ or $\Diamond_{C \leq c} \varphi_2$ holds on some run we keep track of the corresponding point in cost (otherwise the cost value is irrelevant). Every pair or runs gives a pair of outcomes (x_1, x_2) at cost points (c_1, c_2). For every $i = 1 \dots N$

[7] Interleaving intervals for the estimate (even with same confidence) may give non-deterministic results, not to mention that computing estimates is more expensive than hypothesis testing in terms of runs.

[8] This also frees us from the assumption that the processes have some different outputs.

Algorithm 2. Comparison of probabilities

```
function comprise($S$:model , $\psi_1$, $\psi_2$: properties)
$check := 1$, $q := 0$, $t := 0$
while $true$ do
    Observe the random variable $x_1$ corresponding to $\psi_1$ for a run.
    Observe the random variable $x_2$ corresponding to $\psi_2$ for a run.
    if $check = 1$ then
        $x := (x_1 == x_2)$
        $q := q + x * \log(p'_1/p'_0) + (1 - x) * \log((1 - p'_1)/(1 - p'_0))$
        if $q \leq \log(\beta/(1 - \alpha))$ then return $indifferent$
        if $r \geq \log((1 - \beta)/\alpha)$ then $check := 0$
    end
    if $x_1 \neq x_2$ then
        $a := a + c$, $r := r + c$
        if $x_1 = 0$ and $x_2 = 1$ then $t := t + 1$
        if $t \leq a$ then accept process 2.
        if $t \geq r$ then reject process 2.
    end
end
```

we define the new pair of outcomes $(y_{i_1}, y_{i_2}) = \big(x_1 \wedge (i \cdot c/N \geq t_1 \cdot rate_C), x_2 \wedge (i \cdot c/N \geq t_2 \cdot rate_C)\big)$ for which we use our comparison algorithm. We terminate the algorithm when a result for every i^{th} bound is known.

6 Case Studies

We have extended UPPAAL with the algorithms described in this paper. The implementation provides access to all the powerful features of the tool, including user defined functions and types, and use of expressions in guards, invariants, clock-rates as well as delay-rates. Also the implementation supports branching edges with discrete probabilities (using weights), thus supporting probabilistic timed automata (a feature for which our stochastic semantics of NPTA may be easily extended). Besides these additional features, the case-studies reported below (as well as the plots in the previous part of the paper) illustrate the nice features of the new plot composing GUI of the tool[9]. Our objective here is not to study the evolutions of performances with the increase of condidence level, but rather to give a sample of case studies on which our approach can be applied.

Train-Gate Example. We consider the train-gate example [5], where N trains want to cross a one-track bridge. We extend the original model by specifying an arrival rate for Train i ($(i+1)/N$). Trains are then approaching, but they can be stopped before some time threshold. When a train is stopped, it can start again. Eventually trains cross the bridge and go back to their safe state. The template of these trains is given in Fig. 3(a). Our model captures the natural behavior

[9] *http://www.cs.aau.dk/~adavid/smc/* for details.

of arrivals with some exponential rate and random delays chosen with uniform distributions in states labelled with invariants. The tool is used to estimate the probability that Train 0 and Train 5 will cross the bridge in less than 100 units of time. Given a confidence level of 0.05 the confidence intervals returned are $[0.541, 0.641]$ and $[0.944, 1]$. The tool computes for each time bound T the frequency count of runs of length T for which the property holds. Figure 3(b) shows a superposition of both distributions obtained directly with our tool that provides a plot composer for this purpose.

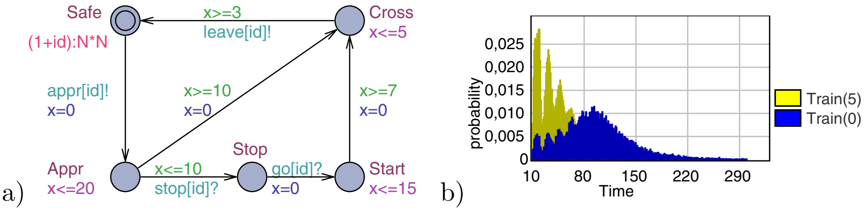


Fig. 3. Template of a train (a) and probability density distributions for $\Diamond_{T \leq t}$`Train(0).Cross` and $\Diamond_{T \leq t}$`Train(5).Cross`

The distribution for Train 5 is the one with higher probability at the beginning, which confirms that this train is indeed the faster one. An interesting point is to note the valleys in the probability densities that correspond to other trains conflicting for crossing the bridge. They are particularly visible for Train 0. The number of valleys corresponds to the number of trains. This is clearly not a trivial distribution (not even uni-modal) that we could not have guessed manually even from such a simple model. In addition, we use the qualitative check to cheaply refine the bounds to $[0.541, 0.59]$ and $[0.97, 1]$.

We then compare the probability for Train 0 to cross when all other trains are stopped with the same probability for Train 5. In the first plot (Fig. 4 top), we check the same property with 100 different time bounds from 10 to 1000 in steps of 10 and we plot the number of runs for each check. These experiments only check for the specified bound, they are not parametrised. In the second plot, we use the parametric extension presented in Section 5 with a granularity of 10 time units. We configured the thresholds u_0 and u_1 to differentiate the comparisons at $u_0 = 1 - \epsilon$ and $u_1 = 1 + \epsilon$ with $\epsilon = 0.1, 0.05, 0.01$ as shown on the figure. In addition, we use a larger time bound to visualise the behaviors after time 600 that are interesting for our checker. In the first plot of Fig. 4, we show for each time bound the average of runs needed by the comparison algorithm repeated 30 times for different values of ϵ. In the bottom plot, we first superpose the cumulative probability for both trains (curves Train 0 and Train 5) that we obtain by applying the quantitative algorithm of Section 4 for each time bound in the sampling. Interestingly, before that point, train 5 is better and later train 0 is better. Second, we compare these probabilities by using the comparison algorithm (curves 0.1 0.05 0.01). This algorithm can retrieve 3 values: 0 if Train 0 wins, 1 if Train 5 wins and 0.5 otherwise. We report for each

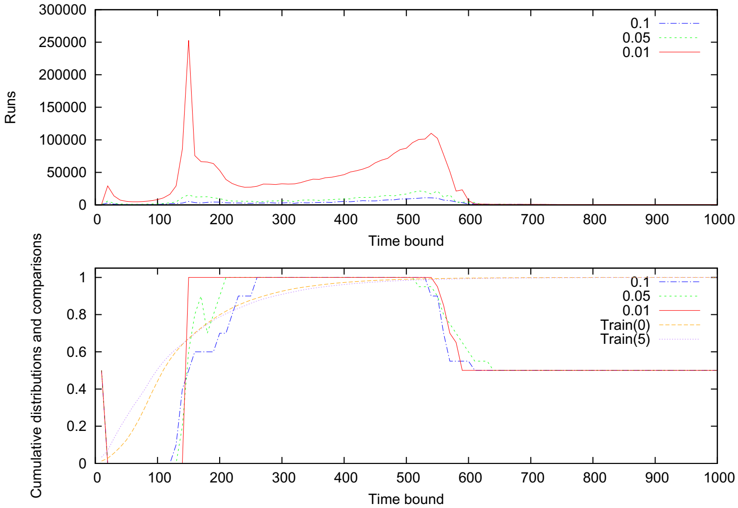


Fig. 4. Comparing trains 0 and 5

time bound and each value of ϵ the average of these values for 30 executions of the algorithm.

In addition, to evaluate the efficiency of computing all results at once to obtain these curves, we measure the accumulated time to check all the 100 properties for the first plot (sequential check), which takes $92s, 182s, 924s$ for $\epsilon = 0.1, 0.05, 0.01$, and the time to obtain all the results at once (parallel check), which takes $5s, 12s, 92s$. The experiments are done on a Pentium D at 2.4GHz and consume very little memory. The parallel check is about 10 times faster[10]. In fact it is limited by the highest number of runs required as shown by the second peak in Fig. 4. The expensive part is to generate the runs so reusing them is important. Note that at the beginning and at the end, our algorithm aborts the comparison of the curves, which is visible as the number of runs is sharply cut.

Lightweight Media Access Control Protocol (LMAC). This protocol is used in sensor networks to schedule communication between nodes. It is targeted for distributed self-configuration, collision avoidance and energy efficiency. In this study we reproduce the improved UPPAAL model from [15] without verification optimisations, parametrise with network topology (ring and chain), add probabilistic weights (exponential and uniform) over discrete delay decisions and examine statistical properties which were not possible to check before. Based on [33], our node model consumes 21, 22, 2 and 1 power units when a node is sending, receiving, listening for messages or being idle respectively.

Fig. 5a shows that collisions may happen in all cases and the probability of collision is higher with exponential decision weights than uniform decision weights, but seems independent of topology (ring or chain). The probability of collision stays stable after 50 time units, despite longer simulations, meaning

[10] The implementation checks simulations sequentially using a single thread.

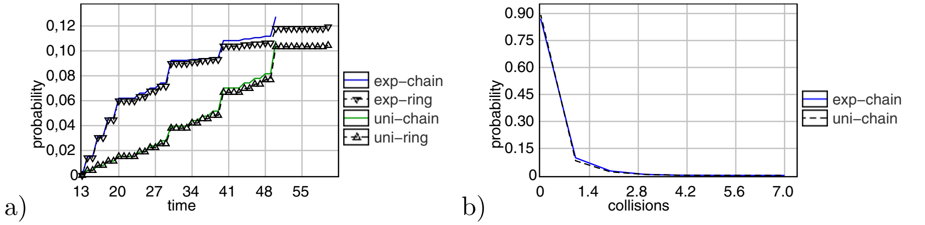


Fig. 5. Collision probabilities when using exponential and uniform weights in chain and ring topologies, a) cumulative probability of collision over time and b) probability of having various numbers of collisions

that the network may stay collision free if the first collisions are avoided. We also applied the method for parametrised probability comparison for the collision probability. The results show that up to 14 time units the probabilities are the same and later exponential weights have higher collision probability than uniform, but the results were inconclusive when comparing different topologies.

The probable collision counts in the chain topology are shown in Fig. 5b, where the case with 0 collisions has a probability of 87.06% and 89.21% when using exponential and uniform weights respectively. The maximum number of probable collisions is 7 for both weight distributions despite very long runs, meaning that the network eventually recovers from collisions.

Fig. 6 shows energy consumption probability density: using uniform and exponential weights in a chain and a ring topologies. The probability `Pr[energy <= 50000](<> time>=1000)` as estimated. Ring topology uses more power (possibly due to collisions), and uniform weights use slightly less energy than exponential weights in these particular topologies.

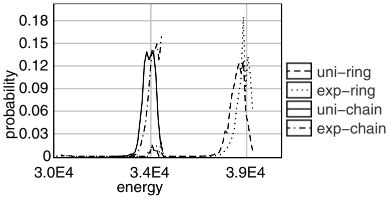


Fig. 6. Total energy consumption

Duration Probabilistic Automata (DPA) [21]. Those automata are used for modelling job-shop problems. A DPA consists of several Simple DPAs (SDPA). An SDPA is a processing unit, a clock and a list of tasks to process sequentially. Each task has an associated duration interval, from which its duration is chosen (uniformly). Resources are used to model task races – we allow different resource types and different quantities of each type. A fixed priority scheduler is used to

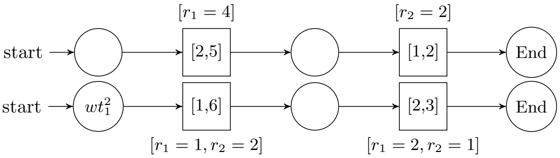


Fig. 7. Rectangles are busy states and circles are for waiting when resources are not available. There are $r_1 = 5$ and $r_2 = 3$ resources available.

resolve conflicts. An example is shown in Fig. 7. DPA can be encoded in our tool (continuous or discrete time semantics) or in Prism (discrete semantics), see the technical report [27]. In Prism, integer and boolean variables are used to encode the current tasks and resources. Prism only supports the discrete time model. In Uppaal, a chain of waiting and task locations is created for each SDPA. Guards and invariants encode the duration of the task, and an array of integers contain the available resources. The scheduler is encoded as a separate template. For Uppaal, we have modelled a discrete version as close as possible to the Prism model ($\mathrm{U}p_p$), an improved discrete version that "jumps" to interesting points in time ($\mathrm{U}p_d$), and a continuous time version that making full use of our formalism ($\mathrm{U}p_c$).

Table 1. Performance of SMC (sec)

Param.			**Estim.**				**Hyp. Testing**			
n	k	m	Prism	$\mathrm{U}p_p$	$\mathrm{U}p_d$	$\mathrm{U}p_c$	Prism	$\mathrm{U}p_p$	$\mathrm{U}p_d$	$\mathrm{U}p_c$
4	4	3	2.7	0.3	0.2	0.2	2.0	0.1	0.1	0.1
6	6	3	7.7	0.6	0.5	0.4	3.9	0.2	0.2	0.3
8	8	3	26.5	1.2	0.9	0.7	16.4	0.5	0.4	0.3
20	40	20	>300				>300	35.5	26.2	20.7
30	40	20	>300				>300	61.2	41.8	33.2
40	40	20	>300				>300	92.2	56.9	59.5
40	20	20	>300				>300	41.1	31.2	26.5
40	30	20	>300				>300	68.8	46.7	46.1
40	55	40	>300				>300			219.5

The performance of the translations is shown in Tab. 1, based on DPAs with n SDPAs, k tasks per SDPA and m resource types. The resource usage and duration interval are randomised. In the hypothesis testing column, Uppaal uses the sequential hypothesis testing introduced in Section 4, whereas Prism uses its own new implementation of the hypothesis testing algorithm. In the estimation column, both Uppaal and Prism use the quantitative check of Section 4, but Uppaal is faster thanks to its more suitable formalism. For both tools, the error bounds used are $\alpha = \beta = 0.05$. In the hypothesis test, the indifference region size is 0.01, while we have $\epsilon = 0.05$ for the quantitative approach. The query for the approximation test is: "What is the probability of all SDPAs ending within t time units?", and for hypothesis testing it is: "Do all SDPAs end within t time units with probability greater than 40%?". The value of t varies for each model as it was computed by simulating the system 369 times and represent the value for which at least 60% of the runs reached the final state. Each number in the table is the average of 10 SMC analyses on the given model. The results show that Uppaal is an order of magnitude faster than Prism even with the discrete encoding, which puts Uppaal at a disadvantage given that it is designed for continuous time[11].

Acknowledgments. We thank Holger Hermans who helped us to clarify the relation between WCTL and the conservative stochastic extension considered in this paper. We are also thankful to David Parker for fruitful discussions on SMC and PRISM.

[11] We note that the number of steps generated for the runs of the PRISM model and the discrete Uppaal model upp_p are comparable.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138(1), 3–34 (1995)
2. Alur, R., Dill, D.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)
3. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) [7], pp. 49–62
4. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T., Größer, M.: Probabilistic and topological semantics for timed automata. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 179–191. Springer, Heidelberg (2007)
5. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
6. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.W.: Minimum-cost reachability for priced timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) [7], pp. 147–161
7. Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.): HSCC 2001. LNCS, vol. 2034. Springer, Heidelberg (2001)
8. Bertrand, N., Bouyer, P., Brihaye, T., Markey, N.: Quantitative model-checking of one-clock timed automata under probabilistic semantics. In: QEST, pp. 55–64. IEEE Computer Society, Los Alamitos (2008)
9. Bogdoll, J., Fiorti, L.-M., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: Bruni, R., Dingel, J. (eds.) FORTE 2011 and FMOODS 2011. LNCS, vol. 6722, pp. 59–74. Springer, Heidelberg (2011)
10. Bohnenkamp, H., D'Argenio, P.R., Hermanns, H., Katoen, J.-P.: Modest: A compositional modeling formalism for real-time and stochastic systems. Technical Report CTIT 04-46, University of Twente (2004)
11. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
12. Clarke, E.M., Donzé, A., Legay, A.: Statistical model checking of mixed-analog circuits with an application to a third order delta-sigma modulator. In: Chockler, H., Hu, A.J. (eds.) HVC 2008. LNCS, vol. 5394, pp. 149–163. Springer, Heidelberg (2009)
13. David, A., Larsen, K.G., Legay, A., Wang, Z., Mikucionis, M.: Time for real statistical model-checking: Statistical model-checking for real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011)
14. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/O automata: a complete specification theory for real-time systems. In: HSCC. ACM, New York (2010)
15. Fehnker, A., van Hoesel, L., Mader, A.: Modelling and verification of the lmac protocol for wireless sensor networks. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 253–272. Springer, Heidelberg (2007)
16. Frehse, G.: Phaver: algorithmic verification of hybrid systems past hytech. STTT 10(3), 263–279 (2008)
17. Gómez, R.: A compositional translation of timed automata with deadlines to uppaal timed automata. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 179–194. Springer, Heidelberg (2009)

18. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004)
19. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004)
20. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009)
21. Kaczmarczyk, V., Sir, M., Bradac, Z.: Stochastic timed automata simulator. In: 4th European Computing Conference. WSEAS
22. Kaczmarczyk, V., Sir, M., Bradac, Z.: Stochastic timed automata simulator. In: Proceedings of the 4th European Computing Conference (2010)
23. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. In: POPL, pp. 344–352 (1989)
24. Maler, O., Larsen, K.G., Krogh, B.H.: On zone-based analysis of duration probabilistic automata. In: INFINITY. EPTCS, vol. 39, pp. 33–46 (2010)
25. Minea, M.: Partial Order Reduction for Verification of Timed Systems. PhD thesis, Carnegie Mellon (1999)
26. Panangaden, P.: Labelled Markov Processes. Imperial College Press (2010)
27. Poulsen, D., van Vliet, J.: Duration probabilistic automata. Technical report, Aalborg University (2011)
28. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 202–215. Springer, Heidelberg (2004)
29. The smv model checker, `http://www.kenmcmil.com/smv.html`
30. The spin tool (spin), `http://spinroot.com/spin/whatispin.html`
31. Teige, T., Eggers, A., Fränzle, M.: Constraint-based analysis of concurrent probabilistic hybrid systems: An application to networked automation systems. In: Nonlinear Analysis: Hybrid Systems (2011)
32. The uppaal tool, `http://www.uppaal.com/`
33. van Hoesel, L.F.W.: Sensors on speaking terms: schedule-based medium access control protocols for wireless sensor networks. PhD thesis, University of Twente (June 2007)
34. Wald, R.: Sequential Analysis. Dove Publisher (2004)
35. Younes, H.L.S.: Verification and Planning for Stochastic Processes with Asynchronous Events. PhD thesis, Carnegie Mellon (2005)
36. Younes, H.L.S.: Ymer: A statistical model checker. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 429–433. Springer, Heidelberg (2005)
37. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002)
38. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to simulink/stateflow verification. In: HSCC, pp. 243–252. ACM, New York (2010)

Robust Model-Checking of Timed Automata via Pumping in Channel Machines

Patricia Bouyer, Nicolas Markey, and Ocan Sankur

LSV, CNRS & ENS Cachan, France

Abstract. Timed automata are governed by a mathematical semantics which assumes perfectly continuous and precise clocks. This requirement is not satisfied by digital hardware on which the models are implemented. In fact, it was shown that the presence of imprecisions, however small they may be, may yield extra behaviours. Therefore correctness proven on the formal model does not imply correctness of the real system.

The problem of robust model-checking was then defined to circumvent this inconsistency. It consists in computing a bound on the imprecision under which the system will be correct.

In this work, we show that robust model-checking against ω-regular properties for timed automata can be reduced to standard model-checking of timed automata, by computing an adequate bound on the imprecision. This yields a new algorithm for robust model-checking of ω-regular properties, which is both optimal and valid for general timed automata.

1 Introduction

Timed automata [1] are a well-established model in real-time system design. These are finite automata augmented with *clocks*, which are used to measure the time elapsed between events, and to constrain the runs of the automaton. Timed automata provide a powerful way of modelling and verifying real-time systems. However, timed automata make idealistic assumptions on the system, such as the perfect continuity of clocks and instantaneous reaction time, which are known not to be preserved in implementation even in digital hardware with arbitrarily small imprecisions. It was shown that even the smallest imprecisions on the clocks yield a different semantics than the *exact* one [14,8] (see Fig. 2 for an example). This suggests that even if the exact semantics is proven correct, the implementation on a physical machine is not guaranteed to respect the specification. In order to prove the correctness of implementations, a framework was proposed in [9], where a detailed model of the implementation of timed automata is given, as programs executed on a simple micro-processor. A simpler over-approximation, the so-called *enlarged semantics* was also studied, which models the imprecisions by *relaxing* all clock constraints of the automaton of the form $x \in [a, b]$ to $x \in [a - \delta, b + \delta]$ for some $\delta > 0$. The problem of *robust model-checking*, that is determining whether for *some* $\delta > 0$, the enlarged semantics satisfies a given property, was first solved for safety properties [14,8], then for linear temporal logic (LTL) [4] (both in PSPACE, which is the complexity of the problem in the exact semantics), and for a timed extension of LTL [5].

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 97–112, 2011.

These robust model-checking algorithms are all valid for a particular class of timed automata, namely, those in which all cycles are *progress cycles*. Roughly, a progress cycle is a cycle of the timed automaton which resets all clocks that are below the maximal constant at least once. We argue that this can be restrictive for modeling. In fact, a timed automaton model of a system under this assumption cannot measure the time spent in a cycle. As an example, consider a simple system which waits for a special signal, while ignoring any other signal, and triggers a time-out action if the expected signal is not received after one second (Fig. [1]). In order to ignore any number of signals during this time, we need a cycle in the automaton. But if all clocks are reset on this cycle, then we cannot measure the time spent in it in order to issue the time-out. One could model such a system using progress cycles by explicitly defining an upper bound m on the number of events that can be treated by the system in one time unit, and unfolding the cycle for m iterations. This would remove the cycle. However this requires the prior knowledge of m which may not be obvious in the design phase, and moreover, this may increase the size of the model and render model-checking infeasible.

```
bool timeout := false;
clock x;
...
x := 0;
while ( x <= 1 ){
   ...
   if ( signal() == A )
      break;
}
if ( x >= 1 )
   timeout := true;
```

Fig. 1. A program that waits for a signal A and issues a time-out if it is not received in one time unit. A timed automaton model of this program naturally contains a non-progress cycle.

Our contribution. We propose a new algorithm for robust model-checking timed automata against ω-regular properties, with optimal complexity (PSPACE). Our algorithm consists in reducing the problem to classical model-checking of timed automata and is valid for general timed automata: we do not assume progress cycles, nor any upper bound on the clocks (Assuming bounded clocks is not restrictive in terms of expressivenes but has a negative effect on the size of the models [2]). We prove that any timed automaton satisfies a given ω-regular property under enlargement by *some* value $\delta > 0$ if, and only if, it satisfies the formula under enlargement by δ_0, where δ_0 only depends on the size of the timed automaton. Then the algorithm simply consists in model-checking the automaton enlarged by δ_0, which can be done using well-known algorithms and tools for timed automata. An algorithm was given in [4] for this problem but only for timed automata with progress cycles and bounded clocks, and because it is based on a modification of the region automaton construction, one cannot use directly the existing model-checking tools. For safety properties, an algorithm similar to ours can be derived from [8], but the complexity would be exponentially higher due to the bound given for δ_0. For automata without nested cycles, [12] gives an algorithm to compute the greatest δ under which a safety property holds, but does not provide a bound on δ.

Although the worst-case complexity of our algorithm is not higher than classical model-checking, in practice, the timed automaton enlarged by δ_0 can yield a model with a state space that is much larger than that of the initial automaton (see Section 3 for the precise value). However, we observed that this does not always increase the time and space necessary for verification. In fact, we used Uppaal [13] to test our algorithm on some benchmarks given with safety specifications.[1] We were able to show the non-robustness of the Fischer protocol upto three agents, and Uppaal returned almost immediately. With more than three agents, the problem is not due to time or space resources but to the fact that Uppaal only allows 32-bit integers as constants in timed automata; when δ_0 requires more precision, the model does not compile. However, Uppaal found counter-examples in less than one minute, for the protocol upto thirty agents, when enlarged by 10^{-8}. We believe that extending Uppaal with arbitrary precision integers, one should be able to use our algorithm for larger models. We could assess the robustness of the CSMA/CD protocol described in [15], the Bang & Olufsen Collision Detection Protocol [11], and the Token Ring Protocol upto thirty agents in less than one minute. Note that the correctness of a model under an enlargement δ implies the correctness for all $0 \leq \delta' < \delta$, so we only verified the above robust models under enlargement that we chose arbitrarily as $\delta = 10^{-6}$ (see [9]). All verification queries returned almost as fast as for the non-enlarged models.

In order to establish our results, we develop proof techniques based on the encoding of the states of timed automata with *channel machines*, introduced in [3], and used in [5] in the context of robustness. In this encoding, a word represents the content of a FIFO channel, which roughly contains all clock symbols ordered by their fractional parts. Time delays are simulated by sequences of read and writes on this channel, whereas action transitions also use a special renaming operation. It turns out that the finitary representation by these words capture well the behaviour of timed automata under enlargement. This was used in [5] to design a robust model-checking algorithm for a timed extension of LTL. We further develop these techniques and prove a pumping lemma for those channel machines, which preserves ω-regular properties. This enables us to prove new properties on the runs of enlarged timed automata, to refine some previously known results and obtain our algorithm. The proof follows the ideas of [14,8] but the techniques are different, and moreover, our analysis is finer since it yields an exponentially better bound for δ_0, as we also noted above.

By lack of space, technical proofs are not included. They can be found in [6].

2 Preliminaries

2.1 Timed Automata

A *labelled timed transition system (LTTS)* is a tuple $(S, s_0, \Sigma, \rightarrow)$, where S is the set of *states*, $s_0 \in S$ the initial state, Σ a finite alphabet, and $\rightarrow \subseteq S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ the *transitions*.

[1] See http://www.uppaal.org/benchmarks/

Given a finite set of clocks $\mathcal{C}$, we call *valuations* the elements of $\mathbb{R}_{\geq 0}^{\mathcal{C}}$. For a subset $R \subseteq \mathcal{C}$, a real $\alpha \in \mathbb{R}_{\geq 0}$ and a valuation v, we write $v[R \leftarrow \alpha]$ for the valuation defined by $v[R \leftarrow \alpha](x) = v(x)$ for $x \in \mathcal{C} \setminus R$ and $v[R \leftarrow \alpha](x) = \alpha$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$, the valuation $v + d$ is defined by $(v+d)(x) = v(x) + d$ for all $x \in \mathcal{C}$. We extend these operations to sets of valuations in the obvious way. We write $\mathbf{0}$ for the valuation which assigns 0 to every clock.

Let $\mathbb{Q}_\infty = \mathbb{Q} \cup \{-\infty, \infty\}$. An *atomic clock formula* is a formula of the form $k \leq x \leq l$ where $x \in \mathcal{C}$ and $k, l \in \mathbb{Q}_\infty$. A *guard* is a conjunction of atomic clock formulas. We denote by $\Phi_{\mathcal{C}}$ the set of guards on the clock set $\mathcal{C}$. We define the *enlargement* of atomic clock constraints by $\delta \in \mathbb{Q}$ as follows: for $x, y \in \mathcal{C}$ and $k, l \in \mathbb{Q}_{>0}$, we let

$$\langle k \leq x \leq l \rangle_\delta \;=\; k - \delta \leq x \leq l + \delta.$$

The enlargement of a guard g, denoted by $\langle g \rangle_\delta$, is obtained by enlarging all its atomic clock constraints. A valuation v *satisfies* a guard g, denoted $v \models g$, if all constraints are satisfied when each $x \in \mathcal{C}$ is replaced by $v(x)$. We denote by $[\![g]\!]$ the set of valuations that satisfy g.

Definition 1. *A timed automaton $\mathcal{A}$ is a tuple $(\mathcal{L}, \mathcal{C}, \Sigma, l_0, E)$, consisting of finite sets $\mathcal{L}$ of* locations*, $\mathcal{C}$ of* clocks*, Σ of* labels*, $E \subseteq \mathcal{L} \times \Phi_{\mathcal{C}} \times \Sigma \times 2^{\mathcal{C}} \times \mathcal{L}$ of* edges*, and where $l_0 \in \mathcal{L}$ is the* initial location*. An edge $e = (l, g, \sigma, R, l')$ is also written as $l \xrightarrow{g,\sigma,R} l'$. Guard g is called the* guard *of e.*

A timed automaton is *integral* if all constants that appear in its guards are integers. For any $\delta \in \mathbb{Q}$, $\mathcal{A}_\delta$ denotes the timed automaton where all guards are enlarged by δ. In the sequel, we only consider integral timed automata as input, and only their enlarged counterparts might not be integral.

Definition 2. *The semantics of a timed automaton $\mathcal{A} = (\mathcal{L}, l_0, \mathcal{C}, \Sigma, E)$ is an LTTS over alphabet Σ, denoted $[\![\mathcal{A}]\!]$, whose state space is $\mathcal{L} \times \mathbb{R}_{\geq 0}^{\mathcal{C}}$. The initial state is $(l_0, \mathbf{0})$.* Delay transitions *are defined as $(l, v) \xrightarrow{\tau} (l, v + \tau)$ for any state (l, v) and $\tau \geq 0$.* Action transitions *are defined as $(l, v) \xrightarrow{\sigma} (l', v')$, for any edge $l \xrightarrow{g,\sigma,R} l'$ in $\mathcal{A}$ such that $v \models g$ and $v' = v[R \leftarrow 0]$.*

Consider any timed automaton $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \Sigma, l_0, E)$ and let $[\![\mathcal{A}]\!] = (S, s_0, \Sigma, \rightarrow)$. A *run* of $[\![\mathcal{A}]\!]$ is a finite or infinite sequence $\rho = (s_i, \sigma_i, \tau_i)_{i \geq 0}$, where $s_i = (l_i, v_i) \in S$, $\sigma_i \in \Sigma$, $\tau_i \in \mathbb{R}_{\geq 0}$ and $s_i \xrightarrow{\tau_i, \sigma_i} s_{i+1}$ for all $i \geq 0$. The word $l_0 l_1 \ldots$ is the *trace* of the run ρ, denoted $\mathsf{trace}(\rho)$. The i-th state s_i of a run ρ is denoted by $(\rho)_i$.

We define the usual notion of *regions* [1]. Pick a timed automaton $\mathcal{A}$ with clock set $\mathcal{C}$, and let M be the largest constant that appears in its guards. For any $(l, u), (l', v) \in \mathcal{L} \times \mathbb{R}_{\geq 0}^{\mathcal{C}}$, we let $(l, u) \simeq (l', v)$ if, and only if, $l = l'$ and for all $x, y \in \mathcal{C}$, the following conditions are satisfied:

- either $\lfloor u(x) \rfloor = \lfloor v(x) \rfloor$ or $u(x), v(x) > M$;
- if $u(x) \leq M$, $\mathsf{frac}(u(x)) = 0$ iff $\mathsf{frac}(v(x)) = 0$;
- if $u(x), u(y) \leq M$, $\mathsf{frac}(u(x)) < \mathsf{frac}(u(y))$ iff $\mathsf{frac}(v(x)) < \mathsf{frac}(v(y))$,

where $\mathsf{frac}(\cdot)$ denotes the fractional part. The equivalence class of a state (l, v) for the relation $\simeq$ is denoted by $\mathsf{reg}((l, v))$, and called a *region* of $\mathcal{A}$. The *region automaton* of $\mathcal{A}$ is a finite automaton $\mathcal{R}(\mathcal{A})$ defined as follows. The states of $\mathcal{R}(\mathcal{A})$ are regions r of $\mathcal{A}$. There is a transition from r to r' labelled by $\sigma \in \Sigma$ if there is an edge (l, g, σ, R, l') such that for some $(l, u) \in r$ and $d \geq 0$, $u + d \models g$, and $(l', u[R \leftarrow 0]) \in r'$. This automaton is known to be time-abstract bisimilar to $[\![\mathcal{A}]\!]$ [1]. The number W of regions is bounded by $|\mathcal{L}| \cdot (2M + 2)^{|\mathcal{C}|} \cdot |\mathcal{C}|! \cdot 2^{|\mathcal{C}|}$. A *progress cycle* in $\mathcal{A}$ is a cycle in $\mathcal{R}(\mathcal{A})$ along which each clock $x \in \mathcal{C}$ is either reset or remains larger than M.

2.2 Robust Model-Checking of Timed Automata

It has been remarked long ago that the semantics of timed automata is not realistic: while this was first exemplified by the so-called *Zeno runs*, the problem goes far beyond, and includes other convergence phenomena [7], or isolated traces [10].

Among the possible approaches to circumvent this problem, *robust model checking* was introduced in [14]: it consists in checking a given property on the extended version of the timed automaton under study; here, *extended* includes clock drifts (clocks may evolve at different rates between $1 - \epsilon$ and $1 + \epsilon$) and guard enlargement. Robust model checking consists in deciding the existence of positive values for ϵ and/or δ for which the property holds in the extended timed automaton. In this paper, we only focus on guard enlargement (*i.e.*, we assume $\epsilon = 0$, so that clocks won't drift); in that setting, robust model checking amounts to deciding the existence of a positive δ for which $\mathcal{A}_\delta$ satisfies a given property.

Take the timed automaton depicted on Fig. 2, and the property that the rightmost location ℓ_3 is never reached. While this property can be checked to hold under the classical semantics, any positive enlargement of the clock constraints will make location ℓ_3 reachable (see [8]); this timed automaton does not *robustly* fulfill the safety property.

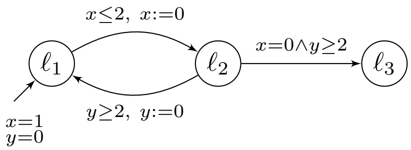


Fig. 2. A (non-robust) timed automaton

Robust model checking has been revisited recently in the setting of *implementability* [9]. Implementability also involves a new semantics for timed automata, the so-called *program semantics*, which simulates the execution of timed automata on a simplified hardware (with digital clock and finite-frequency CPU). This semantics can be over-approximated by the enlarged semantics, so that robust model checking provides an approximate technique to check implementability of timed automata [8].

Robust model checking was proved decidable for safety properties in [14], for timed automata in which all cycles are progress cycles. This was then extended to ω-regular properties [4], and then to timed properties [5].

3 Results

The following theorem is our main result.

Theorem 3. *Let $\mathcal{A}$ be a timed automaton and W be the number of regions of $\mathcal{A}$. Consider any $0 < \delta_0 < \big(8|\mathcal{C}|^2 \cdot (W+1)\big)^{-1}$ if $\mathcal{A}$ has only progress cycles, and $0 < \delta_0 < \big(5(W+1) \cdot |\mathcal{C}|^3 \cdot (2 \cdot |\mathcal{L}| \cdot |\mathcal{C}|! \cdot 4^{|\mathcal{C}|} + 4)^2\big)^{-1}$ otherwise. For any ω-regular property*[2] *ϕ, if $\mathcal{A}_\delta \models \phi$ for some positive δ, then $\mathcal{A}_{\delta_0} \models \phi$.*

Thus, one can decide robust satisfaction of any ω-regular property by checking whether the property holds for some fixed δ_0, which only depends on the size of the automaton. Now, using the usual model-checking algorithms, one can analyze $\mathcal{A}_{\delta_0}$ in polynomial space. In fact, the greatest constant in $\mathcal{A}$ is now multiplied by $\frac{1}{\delta_0}$ and the regions of $\mathcal{A}_{\delta_0}$ can still be encoded in polynomial space. The problem is PSPACE-hard since it is already for timed automata with progress cycles [5].

Corollary 4. *Robust model-checking of general timed automata against ω-regular properties is* PSPACE*-complete.*

The proof of Theorem 3 uses the encoding by channel machines proposed in [5]. The complex mechanism of the channel machine is not required for our purpose. We therefore hide it as much as possible and focus on the underlying transition system. The transition system and its relation to timed automata is presented in section 4. In section 5, we state our main technical results (namely, the pumping lemma and the cycling lemma), which we use to prove Theorem 3. The rest of the paper is then devoted to the proof of these lemmas.

Remark 1. The results of [8] can be lifted to the region-automaton construction, by adding extra transitions representing (progress) cycles [4]. Using our results, this can be further adapted by adding transitions corresponding to weak cycles, which can be detected on the transition system of the channel automaton.

4 Encoding by Channel Machines

In this section, we show how we encode the behaviour of $\mathcal{A}_\delta$ (where $\mathcal{A}$ is a timed automaton and $\delta > 0$) as the transition system of a channel machine. Channel machines are finite-state automata equipped with a FIFO channel. Intuitively, a state of $\mathcal{A}_\delta$ is encoded as follows: the location and the integer parts of the clocks are stored in a *discrete* location, while the channel contains the clock symbols, ordered according to their fractional parts. When a clock is popped out from the tail of the channel, it is (almost) immediately pushed back to the head of the

[2] With *ω-regular property*, we mean state-based properties whose truth value only depends on the set of locations that are visited infinitely often. By an adequate product, we could handle properties expressed by, say, deterministic Muller automata (hence including LTL properties); we omit these details to keep focus on the main objectives of this paper. For an ω-regular property ϕ, we write $\mathcal{A} \models \phi$ when all the runs of automaton $\mathcal{A}$ satisfy ϕ.

channel (hence it is assumed to have small fractional part). This corresponds to a delay transition along which that clock has changed integer value. Some additional symbols (Δ's) will appear on the channel, which serve for refining the region equivalence, and for approximating the values of the clocks. Our encoding is a slightly simplified version of [5], ignoring technicalities such as non-deterministic renaming and occurence testing operations. This is sufficient since the transition system will have access to the whole content of the channel, not only to the head and the queue (as this is the case for the standard mechanism of the channel machines).

We fix for the rest of this section a timed automaton $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \Sigma, l_0, E)$, and a symbol $\Delta \notin \mathcal{C}$.

4.1 Channel Machine Associated to a Timed Automaton

For any word w over alphabet $2^{\mathcal{C}} \setminus \{\emptyset\} \cup \{\Delta\}$, $|w|_\Delta$ denotes the number of occurences of symbol Δ in w, and for any $x \in \mathcal{C}$, $|w|_x$ denotes the number of times x appears inside the symbols of $2^{\mathcal{C}}$ in w. For any integer $N > 0$, let Γ_N be the set of words w over alphabet $2^{\mathcal{C}} \setminus \{\emptyset\} \cup \{\Delta\}$ such that $|w|_\Delta = N$ and $|w|_x \leq 1$ for all $x \in \mathcal{C}$. For any $w \in \Gamma_N$, we define $\mathsf{right}^w_\Delta(x)$ as 0 if $|w|_x = 0$, and as the number of symbols Δ that appears on the right of the (unique) symbol containing x in w. We define $\mathsf{left}^w_\Delta(x)$ symmetrically.

Let M denote the largest constant that appears in $\mathcal{A}$. We assume that clocks are indexed by $\{1, \ldots, n\}$ for some $n > 0$, and we write $\mathcal{C} = \{x_1, \ldots, x_n\}$. We define *the channel machine associated with* $\mathcal{A}$ as the transition system $\mathcal{C}_\mathcal{A}(\Delta^N)$, parameterized by an integer $N \geq 0$, as follows. The states of $\mathcal{C}_\mathcal{A}(\Delta^N)$ are elements of $(\mathcal{L} \times 2^{\mathcal{C}} \times \{0, \ldots, M, \infty\}^{\mathcal{C}}) \times \Gamma_N$. The first component of a state q is the *discrete state*, made of a location, denoted by $\mathsf{loc}(q)$, the set of clocks that have integer values, and a mapping from clocks to their integer parts which is denoted by $\mathsf{int}(q)$ (we write ∞ if it is larger than M); the second component is the *channel content* where clocks are ordered according to their fractional parts. For a state $q = (d, w)$, we extend $\mathsf{right}_\Delta(\cdot)$ as $\mathsf{right}^q_\Delta(x) = \mathsf{right}^w_\Delta(x)$, and similarly for $\mathsf{left}^q_\Delta(x)$. The initial state of $\mathcal{C}_\mathcal{A}(\Delta^N)$ is $((l_0, \mathcal{C}, \mathbf{0}), \Delta^N)$, where l_0 is the initial location of $\mathcal{A}$. Forgetting Δ's, each state q of $\mathcal{C}_\mathcal{A}(\Delta^N)$ naturally encodes a region of $\mathcal{A}$, which we denote by $\mathsf{reg}(q)$: if $\mathcal{C} = \{x, y, z\}$, the state $((l, \{y\}, \left(\begin{smallmatrix}\lfloor x\rfloor=\lfloor y\rfloor=2\\ \lfloor z\rfloor=1\end{smallmatrix}\right)), \Delta^2\{x\}\Delta\{z\}\Delta^4)$ encodes the region where $y = 2$, $\lfloor x \rfloor = 2$, $\lfloor z \rfloor = 1$ and $0 < \mathsf{frac}(x) < \mathsf{frac}(z)$. We will explain the role of the Δ's later.

Transitions of $\mathcal{C}_\mathcal{A}(\Delta^N)$ are labelled by $\Sigma \cup \{\tau\}$, where $\tau \notin \Sigma$. *Elementary delay transitions* are defined as follows, for any state $((l, Z, \iota), w)$:

$$\textit{(i)}\ ((l, Z, \iota), w) \xrightarrow{\tau} ((l, \emptyset, \iota), Z \cdot w) \qquad \textit{(ii)}\ ((l, \emptyset, \iota), w \cdot \Delta) \xrightarrow{\tau} ((l, \emptyset, \iota), \Delta \cdot w)$$

$$\textit{(iii)}\ ((l, \emptyset, \iota), w \cdot X) \xrightarrow{\tau} ((l, X', \iota'), w),\ \text{ where } \iota'(x) = \iota(x) + 1 \text{ for } x \in X,$$
$$\text{and } \iota'(y) = \iota(y) \text{ for } y \notin X,\ \text{ and } X' = X \cap {\iota'}^{-1}([0, M]),$$

where we write $M + 1 = \infty$ (all clocks whose integral part reaches $M + 1$ are abstracted to ∞ and they do not appear anymore in the word of Γ_N—this is the role of ${\iota'}^{-1}([0, M])$). *Delay transitions* are defined as the reflexive and transitive closure of $\xrightarrow{\tau}$, and we also write $\xrightarrow{\tau}$ whenever $\xrightarrow{\tau}^*$. Viewing w as the content of a channel (the head being the first letter and the tail being the last letter), the delay transitions correspond to sequences of reads and writes at the channel, while the discrete state is changed to keep track of the integer parts, whenever a clock subset symbol is read. We say that a clock y *disappears* during a delay transition whenever the rule (iii) is applied, with $y \in X$ and $y \notin X'$. Obviously, delay transitions in $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ correspond to time elapsing in $\mathcal{A}$.

We now define when a state of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ satisfies a guard. A clock formula $x \leq k$ is *exactly satisfied* by a state $q = ((l, Z, \iota), w)$ if either $\iota(x) \leq k-1$, or $\iota(x) = k$ and $x \in Z$ (this is equivalent to say that $\mathsf{reg}(q)$ satisfies $x \leq k$). The formula is *satisfied* if either it is exactly satisfied or $\iota(x) = k$ and $\mathsf{left}^w_\Delta(x) \leq 1$. Intuitively, the value of x is then a bit larger than k (this will be made clearer when explaining the role of the Δ's). A formula $x \geq k$ is exactly satisfied if $\iota(x) \geq k$, and satisfied if it is exactly satisfied or if $\iota(x) = k-1$ and $\mathsf{right}^w_\Delta(x) \leq 1$. *Action transitions* are defined as follows. For any edge $l \xrightarrow{g,\sigma,R} l'$ of $\mathcal{A}$, we let $((l, Z, \iota), w) \xrightarrow{\sigma} ((l, Z \cup R, \iota'), w')$ if $((l, Z, \iota), w)$ satisfies g, $\iota'(x) = 0$ if $x \in R$ and $\iota'(x) = \iota(x)$ if $x \notin R$, and w' is obtained from w by removing the occurences of all clocks in R. This rule is not a valid operation in a channel machine, since some symbols may be removed from w, and checking guards requires reading the tail of w. However this can be simulated using rewriting and occurrence testing, see [5]. Action transitions in $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ where guards are exactly satisfied correspond to action transitions in $\mathcal{A}$. Non-exact satisfaction of guards represents enlarged timing constraints. We will see the precise correspondence later.

A *path* of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ is a sequence $\pi = (q_i, \sigma_i)_{i \geq 1}$ where q_i's are states of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$, and $\sigma_i \in \Sigma \cup \{\tau\}$, and there is a transition labelled by σ_i from q_i to q_{i+1}. We only consider w.l.o.g paths that are alternations of delay and action transitions. The *length* of π, denoted $|\pi|$, is the length of the sequence π. We denote by $\mathsf{trace}(\pi)$ the sequence of locations $\mathsf{loc}(q_0)\mathsf{loc}(q_2)\ldots$ visited by π, and by $\pi_{i\ldots j}$ the path defined by the subsequence between indices i and j. A path is *exact* if all guards in its transitions are satisfied exactly. The i-th state of a path π is denoted by $(\pi)_i$.

Representation of the states of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$. In the sequel, to help manipulate the transition system of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$, we use a *flat representation* of the states. We say that $((l, Z, \iota), \widetilde{w})$ is a *flat representation* of state $(d, w) = ((l, Z, \iota), w)$ whenever $\widetilde{w} \in \left(\mathcal{C} \cup \Delta \cup \Delta^{-1}\right)^*$ can be written as

$$\widetilde{w} = \Delta^{n_0} x_{i_1} \Delta^{n_1} \ldots x_{i_m} \Delta^{n_m}, \tag{1}$$

where $\{x_{i_j} \mid 1 \leq j \leq m\} = \{x \in \mathcal{C} \mid \iota(x) \leq M\}$ is the set of clocks whose integral part is no more than M in (d, w) (some appear in w, some, whose values is integral, do not appear in w), $n_0, \ldots, n_m \geq -1$, and:

- if we remove the maximal prefix of the form $\Delta^{-1} y_1 \Delta^{-1} \ldots \Delta^{-1} y_p$,
- if we remove the maximal suffix of the form $y_p \Delta^{-1} \ldots \Delta^{-1} y_1 \Delta^{-1}$, and
- if we replace all maximal factors of the form $y_1 \Delta^{-1} \ldots \Delta^{-1} y_p$ by $\{y_1, \ldots, y_p\}$,

then we obtain w. Such a flat representation $(d, \widetilde{w})$ contains exactly the same information as w (though with some redundancy) but will be easier to manipulate.

Note that there can be several flat representations for a given state (since both $x\Delta^{-1}y$ and $y\Delta^{-1}x$ can be used to represent $\{x, y\}$). Two clocks x_{i_j} and $x_{i_{j+1}}$ which are separated by Δ^{-1} in $\widetilde{w}$ will belong to the same set in w, hence the corresponding clocks will have the same fractional part in $\mathsf{reg}((d, w))$. If $n_0 = -1$, then x_{i_1} has an integer value in (d, c), and similarly for x_{i_m} if $n_m = -1$. Notice that all clocks whose values are (strictly) less than $M + 1$ are present in this word, even those having an integral value. When clock indices $i_1, \ldots, i_m$ are clear from the context, or implicit, we also represent the channel content (1) by its block sizes $(n_0, n_1, \ldots, n_m)$. The channel content in (1) defines $m + 1$ *blocks*, which are words of $\Delta^* \cup \{\Delta^{-1}\}$ separated by the clock symbols. We enumerate these from 0 to m, and say, for example, that block i has *size* n_i. In the rest, we only use flat representations, for which we easily infer the transition relation.

Example 1. The following is a path in $\mathcal{C}_\mathcal{A}(\Delta^{14})$, for the timed automaton $\mathcal{A}$ depicted on Fig. 2. This path simulates the run of the automaton that enters location ℓ_1 with $x = 1$ and $y = 0$; delays in ℓ_1 for $1 + \delta$ time units, and then moves to ℓ_2, resetting x along that transition. It then waits for $1 - 2\delta$ time units, until $y = 2 - \delta$, and goes back to ℓ_1, and so on.

$$\Big(\big(\ell_1, \{x, y\}, \big(\substack{\lfloor x\rfloor=1\\ \lfloor y\rfloor=0}\big)\big), \Delta^{-1}x\Delta^{-1}y\Delta^{14}\Big) \xrightarrow{\tau} \Big(\big(\ell_1, \emptyset, \big(\substack{\lfloor x\rfloor=2\\ \lfloor y\rfloor=1}\big)\big), \Delta x\Delta^{-1}y\Delta^{13}\Big) \xrightarrow[x:=0]{x\leq 2}$$
$$\Big(\big(\ell_2, \{x\}, \big(\substack{\lfloor x\rfloor=0\\ \lfloor y\rfloor=1}\big)\big), \Delta^{-1}x\Delta y\Delta^{13}\Big) \xrightarrow{\tau} \Big(\big(\ell_2, \emptyset, \big(\substack{\lfloor x\rfloor=0\\ \lfloor y\rfloor=1}\big)\big), \Delta^{12}x\Delta y\Delta\Big) \xrightarrow[y:=0]{y\geq 2}$$
$$\Big(\big(\ell_1, \{y\}, \big(\substack{\lfloor x\rfloor=0\\ \lfloor y\rfloor=0}\big)\big), \Delta^{-1}y\Delta^{12}x\Delta^{2}\Big) \xrightarrow{\tau} \Big(\big(\ell_1, \emptyset, \big(\substack{\lfloor x\rfloor=1\\ \lfloor y\rfloor=2}\big)\big), \Delta x\Delta^{2}y\Delta^{11}\Big) \cdots$$

4.2 Relation with Timed Automata

We now define the relation between $\mathcal{C}_\mathcal{A}(\Delta^N)$ and $\mathcal{A}$, through a relation on their associated time-abstract transition systems. Next, we write $s \overset{\sigma}{\Longrightarrow} s'$ if $s \xrightarrow{\tau} s'' \xrightarrow{\sigma} s'$ for some state s'' (where $\xrightarrow{\tau}$ is a delay transition).

Definition 5. *Let $\mathcal{S} = (S, s_0, \Sigma, \rightarrow)$ be an LTTS. A relation $R \subseteq S \times S$ is a* two-way simulation *if for all $(s_1, s_2) \in R$, if $s_1 \overset{\sigma}{\Longrightarrow} s'_1$ for some $\sigma \in \Sigma$ then $s_2 \overset{\sigma}{\Longrightarrow} s'_2$ for some s'_2 with $(s'_1, s'_2) \in R$, and if $s'_1 \overset{\sigma}{\Longrightarrow} s_1$ for some $\sigma \in \Sigma$, then $s'_2 \overset{\sigma}{\Longrightarrow} s_2$ for some s'_2 with $(s'_1, s'_2) \in R$. A state s_2* simulates *a state s_1 whenever there exists a two-way simulation R such that $(s_1, s_2) \in R$. In that case we write $s_1 \sqsubseteq s_2$.*

For any state (d, w) of $\mathcal{C}_\mathcal{A}(\Delta^N)$ with $w \in \Gamma_N$, we define $\mathsf{concrete}\big((d, w)\big)$ as a subset of $\mathsf{reg}((d, w))$ as follows. It contains a state $(l, v) \in \mathsf{reg}((d, w))$, if, and only if, $l = \mathsf{loc}((d, w))$ and there exists $\delta \in \mathbb{R}^N_{\geq 0}$ that satisfies $0 < \delta(1) < \delta(2) < \ldots < \delta(N) < 1$, $\delta(i + 1) - \delta(i) = \frac{1}{N}$ for all $1 \leq i \leq N - 1$, and $\delta(i) \neq \mathsf{frac}(v(x))$ for all i and $x \in \mathcal{C}$ that appears in w, and, assuming that $\delta(i)$ is the value of the i-th Δ-symbol in w, the extended valuation $v \cup \{\delta(i)\}_{1 \leq i \leq N}$ is ordered according to w.

For example, consider the state $(d, w) = \big((l, \{x\}, ((\substack{\lfloor x\rfloor=\lfloor y\rfloor=0\\ \lfloor z\rfloor=1}))), \Delta^3 z\Delta^3 y\Delta^4\big)$, and valuation v defined by $v(x) = 0$, $v(y) = 0.6$ and $v(z) = 1.3$. We have

$v \in \mathsf{concrete}\big((d,c)\big)$ since for $\delta \in \mathbb{R}^{10}_{\geq 0}$ with $\delta(i) = 0.05 + \frac{i-1}{10}$, the ordering of the fractional parts of $v(x), v(y), v(z), \delta(1), \ldots, \delta(10)$ agree with that given in (d,w).

Lemma 6. *For any timed automaton $\mathcal{A}$ and $N \geq 1$, $[\![\mathcal{A}_{\frac{1}{N}}]\!] \sqsubseteq \mathcal{C}_{\mathcal{A}}(\Delta^N) \sqsubseteq [\![\mathcal{A}_{\frac{2}{N}}]\!]$.*

A weaker version of this lemma was proven in [5]. The two-way simulations in the above lemma are given by relations R defined between states of $[\![\mathcal{A}_\delta]\!]$ and $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ by $(l,v)R(d,w)$ iff $v \in \mathsf{concrete}\big((d,w)\big)$ and $l = \mathsf{loc}(d)$.

4.3 Δ-Distance in $\mathcal{C}_{\mathcal{A}}(\Delta^N)$

If q is a state of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$, we will denote by $[q]$ the topological closure of the region encoded by q. The following lemma characterizes the inclusion of region closures using flat representations and follows from definitions.

Lemma 7. *Let q and q' be two states of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$, and let (d,w) be a flat representation of q, with $w = \Delta^{n_0} x_{i_1} \Delta^{n_1} \ldots x_{i_m} \Delta^{n_m}$. Then $[q] \subseteq [q']$ iff w' has a flat representation of the form (d',w') where $w' = \Delta^{n'_0} x_{i_1} \Delta^{n'_1} \ldots x_{i_m} \Delta^{n'_m}$, s.t.*

- *$\mathsf{loc}(d) = \mathsf{loc}(d')$,*
- *for every $0 \leq i \leq m$, $n'_i = -1$ implies $n_i = -1$, and*
- *there exists $1 \leq r \leq m$ s.t. $n_{r+1} = n_{r+2} = \cdots = n_m = -1$, and:*
  - *for every $1 \leq j \leq r$, $\mathsf{int}(d)(x_{i_j}) = \mathsf{int}(d')(x_{i_j})$,*
  - *for every $r < j \leq m$, $\mathsf{int}(d)(x_{i_j}) = \mathsf{int}(d')(x_{i_j}) + 1$.*

We now define an edit-distance between the states of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$, called the Δ-*distance*. We define the Δ-distance between any pair of states q and q' as infinite unless $[q] \subseteq [q']$ or $[q'] \subseteq [q]$. Fix two flat representations (d,w) and (d',w') that satisfy the conditions in Lemma 7 with block sizes $\boldsymbol{n}$ and $\boldsymbol{n'}$. We define $d_\Delta\big(q,q'\big) = \sum_i (\max({n'_i}^+ - {n_i}^+, 0))$, and notice that this is independent of the choice of the flat representations. This function can be seen to be symmetric (by the fact that both words have the same total number of Δ symbols), and to satisfy the triangular inequality. However, when the function equals 0, this does not imply the equality between states due to the -1-sized blocks. This pseudo-distance has the following important property.

Lemma 8. *For any timed automaton $\mathcal{A}$ and $N \geq |\mathcal{C}| + 2$, for any states q, q' of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$, $d_\Delta(q,q') \leq \frac{N}{|\mathcal{C}|} - 2$ implies that $[q] \cap [q'] \neq \emptyset$.*

5 Proof

5.1 Proof of the Main Theorem

The main theorem is a consequence of the following lemma, using Lemma 6.

Lemma 9. *Let $\mathcal{A}$ be any timed automaton and let W denote its number of regions. Let $K_0 = 2|\mathcal{C}|! \cdot |\mathcal{L}| \cdot 4^{|\mathcal{C}|} + 4$, and $N_1 \geq 8|\mathcal{C}|^2 \cdot (W+1)$ if $\mathcal{A}$ has only progress cycles, and $N_1 \geq 5|\mathcal{C}|^3 \cdot K_0^2 \cdot (W+1)$ otherwise. For any ω-regular property ϕ, if there exists $N > 0$ such that $\mathcal{C}_{\mathcal{A}}(\Delta^N) \models \phi$, then $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1}) \models \phi$.*

We show that if $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1}) \not\models \phi$, then $\mathcal{C}_{\mathcal{A}}(\Delta^{N}) \not\models \phi$ for all $N > 0$. The case where $N < N_1$ is easy, and is implied in the following lemma.

Lemma 10. *For any timed automaton $\mathcal{A}$ and any $N > 0$, $\mathcal{C}_{\mathcal{A}}(\Delta^{N}) \sqsubseteq \mathcal{C}_{\mathcal{A}}(\Delta^{N-1})$.*

The idea is that any path of the channel machine can be carried out when a Δ symbol is removed from the channel. In fact, all guards satisfied in the former system are also satisfied when a Δ symbol is removed. This implies that $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1}) \sqsubseteq \mathcal{C}_{\mathcal{A}}(\Delta^{N})$, hence if the property is violated by a path of $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1})$, then $\mathcal{C}_{\mathcal{A}}(\Delta^{N})$ also has a path violating the property.

In the case where $N_1 < N$, we do not have a simulation between $\mathcal{C}_{\mathcal{A}}(\Delta^{N})$ and $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1})$, but assuming that $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1})$ has a path violating the desired property, we transform it into a path of $\mathcal{C}_{\mathcal{A}}(\Delta^{N})$ also violating the property. This transformation may modify the trace but it does not affect the satisfaction of ϕ (locations that appear infinitely often remain the same). This is stated in the following pumping lemma.

Lemma 11 (Pumping Lemma). *Consider a timed automaton $\mathcal{A}$, and let W denote its number of regions. Let $K_0 = 2|\mathcal{C}|!\cdot|\mathcal{L}|\cdot 4^{|\mathcal{C}|}+4$, and $N_1 \geq 8|\mathcal{C}|^2 \cdot (W+1)$ if $\mathcal{A}$ has only progress cycles, and $N_1 \geq 5|\mathcal{C}|^3 \cdot K_0^2 \cdot (W+1)$ otherwise. Then, for any path π of $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1})$, for any $L \geq 0$, there exists a path π' of $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1+L})$, such that the same set of locations appear infinitely often in π and π'.*

The rest of the paper is devoted to the proof of the pumping lemma. The path of $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1+L})$ is obtained by repeating some factors of the path of $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1})$, then repeating some factors of the resulting word, and repeating this a finite number of times. This operation preserves ω-regular properties.

Overview of the proof. We start by studying, in Subsection 5.2, how the sizes of the blocks evolve along a path. We characterize the blocks whose sizes do not become small along a path; these are called the blocks that *stay united*. We prove a pumping lemma for these blocks (Lemma 12). Then, we study, in Subsection 5.3, exact paths of $\mathcal{C}_{\mathcal{A}}(\Delta^{N_1})$ and show that for any path of bounded length, there is an exact path that follows the same trace and that is close in terms of Δ-distance (Lemma 14). In Subsection 5.4, we apply the above results to bounded paths to prove the pumping lemma for unbounded paths.

5.2 Pumping Lemma: Bounded Case

We fix a timed automaton $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \Sigma, l_0, E)$ and $N > 0$. Let W denote the number of regions of $\mathcal{A}$. For any channel content $w \in \Gamma_N$ with clocks $x_{i_1}, \ldots, x_{i_m}$ and block sizes $(n_0, \ldots, n_m)$, and $L \geq 0$, we define $w[x_{i_j} \leftarrow x_{i_j}\Delta^L]$ as the word of Γ_{N+L} obtained from w by replacing the j-th block Δ^{n_j} with $\Delta^{n_j^+ + L}$. We assume that i_0 is an index fixed to 0. We extend the above definition to the 0-th block, by writing $w[x_{i_0} \leftarrow x_{i_0}\Delta^L] = w[x_{i_1} \leftarrow \Delta^L x_{i_1}]$, obtained by inserting L new Δ symbols in the 0-th block.

We fix the constant $N_0 = 2W + 2$. A block is *small* if it has size $\{-1, 0, 1\}$, *medium* if it has size $\{2, \ldots, N_0 - 1\}$, and *large* otherwise. An important observation about $\mathcal{C}_{\mathcal{A}}(\Delta^{N})$ is that, if a guard is satisfied at some state (d, w), then, when an arbitrary number of Δ symbols are inserted in medium/large blocks, the same guard is still satisfied. However, if we insert additional Δ symbols inside small blocks, then formulas which are satisfied but not exactly satisfied may not be satisfied anymore. We define a notion of *staying united* along a path for blocks. Intuitively, such blocks are those that are either always at least medium, or are cut into at least one medium/large block along the path. We then show that one can insert any number of Δ symbols inside a block that stays united, and adapt the original path of $\mathcal{C}_\mathcal{A}(\Delta^N)$ to a path in $\mathcal{C}_\mathcal{A}(\Delta^{N+L})$.

We define a relation on pairs of states and clock indices as follows. Let $q = (d, w)$ and $q' = (d', w')$ denote two flat representations of states of $\mathcal{C}_\mathcal{A}(\Delta^N)$ such that $q \xrightarrow{\tau} q'$ is an elementary delay transition. We let $(q, i) \prec (q', j)$ whenever for every integer $L > 0$, there is a delay transition $q[x_i \leftarrow x_i\Delta^L] \xrightarrow{\tau} q'[x_j \leftarrow x_j\Delta^L]$. This relation can be characterized rather easily by analysing all possible cases for elementary delays $q \xrightarrow{\tau} q'$. This relation is extended to the transitive closure of delay transitions. Similarly, assuming $q \xrightarrow{\sigma} q'$ is an action transition, we write $(q, i) \prec (q', j)$ whenever for every integer L, there is an action transition $q[x_i \leftarrow x_i\Delta^L] \xrightarrow{\sigma} q'[x_j \leftarrow x_j\Delta^L]$. This can be characterized easily as well.

For any finite path π in $\mathcal{C}_\mathcal{A}(\Delta^N)$, and any block i_1 in $(\pi)_1$, we say that *block* i_1 *stays united* in π, if block i_1 is large in $(\pi)_1$, and if there exists clock indices $i_2, \ldots, i_m$ such that $((\pi)_1, i_1) \prec ((\pi)_2, i_2) \prec \ldots \prec ((\pi)_n, i_n)$, with $n = |\pi|$.

By definition, the paths along which a block stays united are not sensitive to the precise size of those blocks. This is formalized in the following lemma, which is a pumping lemma for particular finite paths.

Lemma 12. *Let π be a path of $\mathcal{C}_\mathcal{A}(\Delta^N)$ such that $((\pi)_1, i_1) \prec ((\pi)_2, i_2) \prec \ldots \prec ((\pi)_n, i_n)$, for some $n > 0$. Then for any $L > 0$, $\mathcal{C}_\mathcal{A}(\Delta^{N+L})$ has a path π' with $(\pi')_j = (\pi)_j[x_{i_j} \leftarrow x_{i_j}\Delta^L]$ for any $1 \leq j \leq n$ and $\mathsf{trace}(\pi) = \mathsf{trace}(\pi')$.*

We now state a lower bound on the length of a path along which a block does not stay united. The idea is that if a block does not stay united, then whenever it is cut in two parts during a transition, either one of the resulting blocks is small, or none of them stays united in the rest of the path.

Lemma 13. *Let π be a path of $\mathcal{C}_\mathcal{A}(\Delta^N)$ with $|\pi| = p$. Then all blocks in $(\pi)_1$ that have size at least $p+1$ stay united along π.*

5.3 Making Exact Paths

In this section, we show how to transform an arbitrary path of bounded length of $\mathcal{C}_\mathcal{A}(\Delta^N)$ into an exact one. By definition, if a path is not exact, then there are states with small blocks. The idea of our transformation is to replace all small blocks and the blocks that do not stay united by -1-sized blocks, while preserving the ordering of the clocks. Notice that by Lemma 7, the states obtained by this operation define closed subregions of those defined by the original states. Clearly, if all small blocks have size -1, then any guard that is satisfied by a state is satisfied exactly, so the new path is exact.

Take $N \geq (|\mathcal{C}|+1) \cdot N_0$, and fix a path π of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ of length $n \leq N_0 - 1$. For each $1 \leq i < i' \leq n$, we associate with $(\pi)_i$ a state $H(\pi, i, i')$ where any small or medium block that does not stay united along $\pi_{i\dots i'}$ is replaced by blocks of size -1. Formally, let $j_1, \dots, j_k$ denote the indices of the blocks of $(\pi)_i$ that do not stay united along the path $\pi_{i\dots i'}$. Let us write $(\pi)_i = (d, \boldsymbol{n})$. We define $H(\pi, i, i') = (d', \boldsymbol{n}')$ as follows. We let $n'_{j_1} = \dots = n'_{j_k} = -1$, and $n'_{j_0} = n_{j_0} + n^+_{j_1} \dots + n^+_{j_k}$, for the large block with the minimal index j_0, which exists by the choice of N. (Notice that the closed region $[H(\pi, i, i')]$ is independent of j_0). We have, by Lemma 7, $[H(\pi, i, i')] \subseteq [(\pi)_i]$. The same lemma implies that any state q with $[q] \subseteq [H(\pi, i, i')]$, has only blocks that stay united along $\pi_{i\dots i'}$. Observe that because $(\pi)_i$ has at least one large block, by Lemma 13, $H(\pi, i, i')$ is well-defined. Last, we have $d_\Delta((\pi)_i, H(\pi, i, i')) \leq |\mathcal{C}| \cdot N_0$ by construction.

We construct exact paths that are "close" to the original ones, as follows.

Lemma 14. *Let $\mathcal{A}$ be any timed automaton having only progress cycles, and $N \geq (|\mathcal{C}|+1) \cdot N_0$. Let π a path of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ of length at most N_0. Then, there exists an exact path π' of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ over trace* $\mathsf{trace}(\pi_{1\dots N_0})$*, with $(\pi')_1 = H(\pi, 1, N_0)$ and $[(\pi')_i] \subseteq [H(\pi, i, N_0)]$ and $d_\Delta(\pi, \pi') \leq (|\mathcal{C}|+1)N_0$ for all $1 \leq i \leq N_0$.*

A result similar to Lemma 14 was given in [8, Th. 44] for runs of timed automata and distance d_∞ over valuations. The proof there involved approximation of the width of parametric DBMs. Our approach is in some sense closer to the input timed automaton, which may explain why we get an exponentially better distance to the original run.

As one might expect, exact paths satisfy the following property. The idea is that exact paths of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ are not sensitive to the sizes of the blocks.

Lemma 15. *Let $\mathcal{A}$ be any timed automaton, $N \geq 1$ and π an exact path of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$. Then, for any $N' \geq N$, and any state q of $\mathcal{C}_{\mathcal{A}}(\Delta^{N'})$ with $[q] \subseteq [\mathsf{first}(\pi)]$, there exists an exact path π' over the same trace as π, with $\mathsf{first}(\pi') = q$ and $[(\pi')_i] \subseteq [(\pi)_i]$ for all $1 \leq i \leq |\pi|$. The same property holds backwards: for any $q \in [\mathsf{last}(\pi)]$, there exists an exact path π' over the trace of π in $\mathcal{C}_{\mathcal{A}}(\Delta^{N'})$ with $\mathsf{last}(\pi') = q$ and $[(\pi')_i] \subseteq [(\pi)_i]$ for $1 \leq i \leq |\pi|$.*

5.4 Pumping Lemma with Progress Cycles: Unbounded case

The previous sections dealt with the properties of the bounded paths of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$. We now use these to prove the pumping lemma for infinite paths. Let us first define a transformation on the traces of the runs. For any finite trace $w \in \mathcal{L}^*$, we let $\widetilde{w} = \{u_1^+ u_2^+ \dots u_n^+ \mid u_1 u_2 \dots u_n = w\}$.

We first need the following lemma, which is an adaptation of Lemma 29 of [8] to channel machines.

Lemma 16 (Cycling Lemma). *Let $\mathcal{A}$ be any timed automaton, $N \geq 1$ and π an exact progress cycle in $\mathcal{C}_{\mathcal{A}}(\Delta^N)$. Then, for all states q with $[q] \subseteq [\mathsf{last}(\pi)]$, there exists a path π' in $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ with $\mathsf{first}(\pi') = \mathsf{first}(\pi)$ and $[\mathsf{last}(\pi')] \subseteq [q]$, and $\mathsf{trace}(\pi') \in \widetilde{\mathsf{trace}(\pi)}$.*

We are now ready to prove the pumping lemma, for timed automata with progress cycles. Figure 3 illustrates a step of the proof.

Proof (of Lemma 11). We prove the result for $1 \leq L \leq |\mathcal{C}|N_0 - 2$. For larger L, one can repeat this construction. Let $N \geq 4|\mathcal{C}|^2 N_0$, and consider an infinite path π of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ where $\mathsf{first}(\pi)$ is the initial state of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$.

Let $n = N_0 - 1$. Let $G^L(\pi, i, i+n)$ denote the state of $\mathcal{C}_{\mathcal{A}}(\Delta^{N+L})$ obtained from $(\pi)_i$ by inserting Δ^L in the block with minimal index that stays united along $\pi_{i \ldots i+n}$ (such a block exists by Lemma 13). We also define $H^L(\pi, i, i+n)$ by inserting Δ^L to the same block in $H(\pi, i, i+n)$ ($H(\cdot)$ is defined right before Lemma 14). Then, by construction, $d_\Delta\big(H^L(\pi, i, i+n), G^L(\pi, i, i+n)\big) \leq |\mathcal{C}|N_0$.

We now define a path π' of $\mathcal{C}_{\mathcal{A}}(\Delta^{N+L})$ over trace $\mathsf{trace}(\pi)$. At each step $i \geq 1$, we construct $\pi'_{\beta_i \ldots \beta_{i+1}}$, where $(\beta_i)_{i \geq 1}$ is an increasing sequence. Our construction satisfies $\mathsf{trace}(\pi'_{\beta_i \ldots \beta_{i+1}}) \in \mathsf{trace}(\pi_{\alpha_i \ldots \alpha_{i+1}})$, and $[(\pi')_{\beta_i}] \subseteq [(\pi)_{\alpha_i}]$, for some possibly different increasing sequence $(\alpha_i)_{i \geq 1}$.

We define $(\pi')_1$ as the initial state of $\mathcal{C}_{\mathcal{A}}(\Delta^{N+L})$, which satisfies $[(\pi')_1] \subseteq [(\pi)_1]$. Suppose now that $\pi'_{1 \ldots \beta_i}$ has been constructed for some $\beta_i \geq 1$. By Lemma 12, there is a path g of $\mathcal{C}_{\mathcal{A}}(\Delta^{N+L})$ from $G(\pi, \alpha_i, \alpha_i + n)$ over $\mathsf{trace}(\pi_{\alpha_i \ldots \alpha_i + n})$, such that $(g)_j = (\pi)_{\alpha_i + j}[z_j \leftarrow z_j \Delta^L]$ for some clocks z_j. We then apply Lemma 14 to g to get an exact path h with $(h)_1 = H^L(\pi, \alpha_i, \alpha_i + n)$ over the trace of g, with $d_\Delta\big((h)_j, (g)_j\big) \leq (|\mathcal{C}| + 1)N_0$, and $[(h)_j] \subseteq [H^L(\pi, \alpha_i + j, \alpha_i + n)]$ for all $1 \leq j \leq n$. Now, h contains at least $n/2 \geq W$ action transitions, so there exist $1 \leq l_0 < l_1 \leq n$ such that $\mathsf{reg}((h)_{l_0}), \mathsf{reg}((h)_{l_0+1}), \ldots, \mathsf{reg}((h)_{l_1})$ is a progress cycle of the region automaton of $\mathcal{A}$. We have $d_\Delta\big((h)_{l_1}, H^L(\pi, \alpha_i + l_1, \alpha_i + l_1 + n)\big) \leq (3|\mathcal{C}| + 1)N_0 - 2$ by combining the following inequalities:

$$
\begin{aligned}
&d_\Delta((h)_{l_1}, (g)_{l_1}) \leq (|\mathcal{C}| + 1)N_0, \\
&d_\Delta\big((g)_{l_1}, G^L(\pi, \alpha_i + l_1, \alpha_i + l_1 + n)\big) \leq L \leq |\mathcal{C}|N_0 - 2, \\
&d_\Delta\big(G^L(\pi, \alpha_i + l_1, \alpha_i + l_1 + n), H^L(\pi, \alpha_i + l_1, \alpha_i + l_1 + n)\big) \leq |\mathcal{C}|N_0.
\end{aligned}
$$

By Lemma 15, $\mathcal{C}_{\mathcal{A}}(\Delta^{N+L})$ has a path over $\mathsf{trace}(h)$ from $(\pi')_{\beta_i}$ to some state q with $[q] \subseteq [(h)_{l_1}]$. By Lemma 8, we get $[(h)_{l_1}] \cap [H^L(\pi, \alpha_i + l_1, \alpha_i + l_1 + n)] \neq \emptyset$.

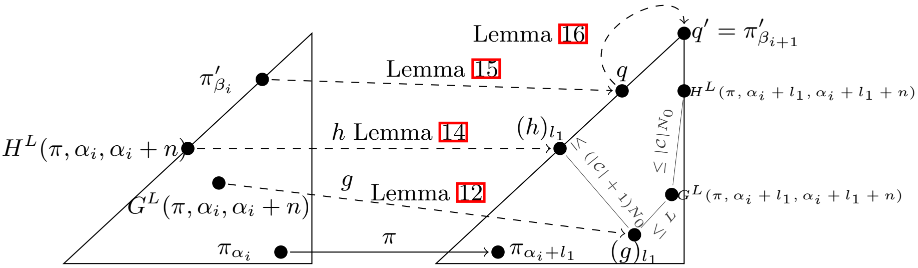

Lemma 16
$q' = \pi'_{\beta_{i+1}}$
π'_{β_i}
Lemma 15
q
$H^L(\pi, \alpha_i + l_1, \alpha_i + l_1 + n)$
$H^L(\pi, \alpha_i, \alpha_i + n)$
h Lemma 14
$(h)_{l_1}$
$\leq |\mathcal{C}|N_0$
$\leq (|\mathcal{C}| + 1)N_0$
g
Lemma 12
$G^L(\pi, \alpha_i + l_1, \alpha_i + l_1 + n)$
$G^L(\pi, \alpha_i, \alpha_i + n)$
$\leq L$
π_{α_i}
π
$\pi_{\alpha_i + l_1}$
$(g)_{l_1}$

Fig. 3. An induction step of the proof of Lemma 11. Two triangles represent two closed regions. Their sides, interiors and corners are subregions. The proof constructs the dashed paths bottom-up.

Lemma 16 provides a path of $\mathcal{C}_{\mathcal{A}}(\Delta^{N+L})$ from q to some state q' with $[q'] \subseteq [(h)_{l_1}] \cap [H^L(\pi, \alpha_i + l_1, \alpha_i + l_1 + n)]$, over a trace in $\mathsf{trace}(h_{l_0 \ldots l_1})^+$. The concatenation of these two paths define $\pi'_{\beta_i..\beta_{i+1}}$. This concludes a step of the induction. □

5.5 Pumping Lemma with Non-progress Cycles

In this subsection, we explain the generalization of the proof of the pumping lemma to the case of timed automata with non-progress cycles. Let us call *weak cycle*, a path π of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$ with $[\mathsf{last}(\pi)] \subseteq [\mathsf{first}(\pi)]$ that is not a progress cycle. Thus, π is a cycle along which at least one clock that is present on the channel is not reset. We show that all weak cycles can be transformed into *weak quasi-exact cycles* (defined below). We then define *quasi-exact paths* of $\mathcal{C}_{\mathcal{A}}(\Delta^N)$, which are paths that are exact except along weak quasi-exact cycles. Quasi-exact paths behave very much like exact paths: We adapt each of the previous lemmas involving exact paths for quasi-exact paths, and the proof in the presence of weak cycles is very similar to the proof which assumes progress cycles.

Example 2. The following path of $\mathcal{C}_{\mathcal{A}}(\Delta^{10})$ is an example of weak cycle, in which clock z is not reset.

$$(\ell_1, \{x\}, \cdot), \Delta^3 y \Delta^4 z \Delta^3 \xrightarrow{\tau} (\ell_1, \emptyset, \cdot), \Delta^2 x \Delta^3 y \Delta^4 z \Delta \xrightarrow{\sigma}$$
$$(\ell_2, \{y\}, \cdot), \Delta^2 x \Delta^7 z \Delta \xrightarrow{\tau} (\ell_2, \emptyset, \cdot), y \Delta^2 x \Delta^7 z \Delta \xrightarrow{\sigma'} (\ell_1, \{x\}, \cdot), y \Delta^9 z \Delta$$

It can be seen on this example how clock z prevents from accessing the Δ's that accumulate immediately on its left.

Given a weak cycle, clocks that are reset along the cycle are called *active*, and others *inactive*. Consider a weak cycle π. Suppose that $A = \{i_1, \ldots, i_r\}$ are the indices of the active clocks in π, given in the order of their appearance in $\mathsf{last}(\pi)$. Then, π can be factorized as,

$$\pi = \pi_{i_r} \pi_{i_{r-1}} \ldots \pi_{i_1} \pi', \qquad (2)$$

where π_{i_j} ends with the last reset of the clock x_{i_j} in π, and no clock is reset in π'. An important observation that we use is that the size of the block i_j in $\mathsf{last}(\pi_{i_j})$ is determined by the number of Δ's read inside π_{i_j}, for any $1 \leq j \leq r-1$. The block i_r is particular, since its size at the last state is the sum of all the blocks $i_1, \ldots, i_r$ in $\mathsf{first}(\pi)$, plus the Δ's read during π_{i_r}. Among the blocks A of $\mathsf{last}(\pi)$, some will be called *pumpable*. Formally, for $K_0 = |\mathcal{L}| \cdot |\mathcal{C}|! \cdot 4^{|\mathcal{C}|+1} + 4$, we let

$$\mathsf{Pumpable}(\pi) = \{i_j \in A \mid \exists \tau \in \pi_{i_j}, \mathsf{time}_\Delta(\tau) \geq 2 \text{ or } \mathsf{time}_\Delta(\pi_{i_j}) \geq K_0\},$$

where $\mathsf{time}_\Delta(\tau)$ denotes the number of Δ's read from the channel in the delays of a given path τ, and with the abusive notation $\exists \tau \in \pi_{i_j}$ meaning "*for some delay transition in π_{i_j}*". We prove a pumping lemma for pumpable blocks inside weak cycles. In fact, if π_{i_j} has a delay transition which reads at least two Δ symbols, then block i_0 becomes medium or large during this delay, so it can be extended to read more Δ symbols. But if all delay transitions are too short then this trick cannot be used; in that case when a large number of transitions occur inside π_{i_j}, we show that some factor can be repeated, while additional delays that read Δ's are inserted (this is why we need a large constant K_0).

We then define *quasi-exact paths*, which are paths that are made of delays, exact transitions and weak cycles in which any block that is active is either pumpable, or it ends with size -1. We show that these paths behave very much like exact paths, and we follow step-by-step the same lemmas to construct the proof of the pumping lemma in the general case.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Bouyer, P., Chevalier, F.: On conciseness of extensions of timed automata. Journal of Automata, Languages and Combinatorics 10(4), 393–405 (2005)
3. Bouyer, P., Markey, N., Ouaknine, J., Worrell, J.: The cost of punctuality. In: LICS 2007, Wrocław, Poland, pp. 109–118. IEEE Computer Society Press, Los Alamitos (2007)
4. Bouyer, P., Markey, N., Reynier, P.-A.: Robust model-checking of linear-time properties in timed automata. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 238–249. Springer, Heidelberg (2006)
5. Bouyer, P., Markey, N., Reynier, P.-A.: Robust analysis of timed automata via channel machines. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 157–171. Springer, Heidelberg (2008)
6. Bouyer, P., Markey, N., Sankur, O.: Robust model-checking of timed automata via pumping in channel machines. Research Report LSV-11-19, Laboratoire Spécification et Vérification, ENS Cachan, France (2011)
7. Cassez, F., Henzinger, T.A., Raskin, J.-F.: A comparison of control problems for timed and hybrid systems. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 134–148. Springer, Heidelberg (2002)
8. De Wulf, M., Doyen, L., Markey, N., Raskin, J.-F.: Robust safety of timed automata. Formal Methods in System Design 33(1-3), 45–84 (2008)
9. De Wulf, M., Doyen, L., Raskin, J.-F.: Almost ASAP semantics: From timed models to timed implementations. Formal Aspects of Comput. 17(3), 319–341 (2005)
10. Gupta, V., Henzinger, T.A., Jagadeesan, R.: Robust timed automata. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 331–345. Springer, Heidelberg (1997)
11. Havelund, K., Skou, A., Larsen, K.G., Lund, K.: Formal modeling and analysis of an audio/video protocol: an industrial case study using uppaal. In: RTSS 1997. IEEE Computer Society, Los Alamitos (1997)
12. Jaubert, R., Reynier, P.-A.: Quantitative robustness analysis of flat timed automata. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 229–244. Springer, Heidelberg (2011)
13. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. Int. Journal on Software Tools for Technology Transfer 1, 134–152 (1997)
14. Puri, A.: Dynamical properties of timed systems. Discrete Event Dynamic Systems 10(1-2), 87–113 (2000)
15. Yovine, S.: Kronos: A verification tool for real-time systems. International Journal on Software Tools for Technology Transfer 1, 123–133 (1997)

Thin and Thick Timed Regular Languages

Nicolas Basset[1] and Eugene Asarin[2]

[1] LIGM, Université Paris-Est and CNRS
nbasset@dptinfo.ens-cachan.fr
[2] LIAFA, Université Paris Diderot and CNRS
asarin@liafa.jussieu.fr

Abstract. In previous literature on timed automata, it was noticed that they are in several aspects too precise, which leads sometimes to strange artifacts, mathematical pathologies or unrealistic models. In particular, some timed automata are non-implementable, non-robust, behave badly under discretization, have many Zeno runs etc. In this paper, we propose a unifying approach to most of these issues for deterministic timed automata. We classify these automata either as **thin** or as **thick**. In thin automata, all the infinite trajectories are, in some weak sense, Zeno; the discretization of long trajectories is difficult, since it requires very small discretization step. In thick automata, most of trajectories are non-Zeno and behave well under discretization; such automata satisfy a sort of pumping lemma. Formally, the thin-thick alternative is based on the notion of entropy of timed regular languages introduced by E. Asarin and A. Degorre in [3,4]. Thin languages have the entropy $= -\infty$ while thick have a larger one. An important application of thin-thick alternative is again the entropy theory of timed languages. We show that the entropy can be computed with a desired precision using discretization and thus it is computable, which closes a question left open in [3,4].

1 Introduction

Timed automata [2] using exact continuous clocks, exact guards and resets are a beautiful mathematical object and a useful model of real-time systems. However, from the very beginning of the timed automata research, it was clear that they are in several aspects too precise, which leads sometimes to strange artifacts, mathematical pathologies or unrealistic models. Several lines of research have partially elucidated these issues.

Thus, the state space of a timed automaton being infinite, some long (or infinite) runs never revisit the same state. For this reason, as stated in [7], usual **pumping lemmata** do not hold, and should be replaced by rather involved analogues. In a run, infinitely many events can happen during a finite amount of time, or two events can happen again and again with the time interval between them tending to 0. Such a run reminds of Zeno's aporias and is often called a **Zeno run**, see [12] and reference therein. Pathological runs do not support well discretization of clocks, see [14,6].

In order to rule out bad behaviors, restricted classes of timed automata and alternative semantics were considered by several authors. Thus, in [13,15], a **tube**

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 113–128, 2011.

language semantics is introduced. In [19] a **robust semantics**, based on small imprecisions is considered. It reappears in a different flavor as **implementability**, see [22,21], and in another version in [1].

With the same objective to rule out bad behaviors, restrictions are often put on all the cycles in the automaton, by requiring that each cycle takes at least one time unit (strongly non-Zeno condition), or resets all the clocks (progress cycle condition), or even resets all the clocks at one and the same transition (regeneration or synchronization condition).

In this paper, we propose a unifying approach to most of these issues for deterministic timed automata. We classify each automaton either as *thin* or as *thick* (the classification is decidable).

In thin automata. all the infinite trajectories, are, in some weak sense Zeno; the digitization of long trajectories is difficult, since it requires very small discretization step.

In thick automata. most of trajectories are non-Zeno, behave well under digitization and satisfy a sort of pumping lemma.

The main technical tool used to characterize thin and thick timed languages is their (volumetric) entropy introduced in [3,4]. Let us briefly recall this notion: a timed language L, for a given number of events n, can be seen as several polytopes in $\mathbb{R}^n$, their total volume is denoted by V_n. In most cases, for $n \to \infty$ this volume behaves exponentially: $V_n \approx 2^{n\mathcal{H}}$. The growth rate (i.e. $\mathcal{H}$) is referred to as *entropy*, and characterizes the size and the information contents of the timed language. With the notion of entropy, the definition of thin and thick languages is simple: a language is thin if its entropy equals $-\infty$ (that is the volume V_n decays faster than any exponent), and thick otherwise.

We identify a novel notion of a forgetful cycle, that is a cyclic path allowing forgetting the clock values. We state that any path in a timed automaton which is thick and long (in some precise sense), necessarily contains a forgetful cycle (Thm. 1), which can be seen as a weak version of pumping lemma. Based on this pumping lemma, we obtain our first main result (Thm. 2): thickness of a language is equivalent to many other nice properties briefly described above (good discretization, existence of forgetful cycle etc.).

The proof of Thms. 1-2 is rather technical, and uses together with "timed" techniques inspired by [19,1], the monoid version of Ramsey theory, namely Simon factorization forests [20].

The thin-thick alternative leads to a more precise analysis of a timed automaton when applied to its strongly connected components. In general, a timed automaton can be decomposed into several strongly connected components (some of them are thin, others thick) and acyclic pathways between them. We show that most of the long enough runs spend most of the time in thick components, and only few pathological runs wander in thin components (Thm. 3).

Finally, we apply the thin-thick alternative to the analysis of entropy of timed languages. In [4] it was shown that the entropy of a timed language can be lower and upper bounded using entropies of two discrete languages L_ε^- and L_ε^+, corresponding to a shrunk and a bloated discretizations of L:

$$h(L_\varepsilon^-) + \log(\varepsilon) \leq \mathcal{H} \leq h(L_\varepsilon^+) + \log(\varepsilon).$$

Here (Thm. 4) we strengthen this result and establish that on a thick component the entropy of a timed language can be approximated with a good precision using the entropy $h(L_\varepsilon)$ of the discretized language:

$$\mathcal{H} = h(L_\varepsilon) + \log(\varepsilon) + o(1).$$

As a corollary we obtain a converging algorithm allowing computation of $\mathcal{H}$ for any timed regular language with any precision required. This answers the open question from [4]: $\mathcal{H}$ is always a computable real number.

Paper organization. In Sect. 2, we recall some basic definitions, define thin and thick languages and give some motivating examples. In Sect. 3 we describe three more involved constructions: polytopes associated to paths in timed automata as in [18], region split automaton as in [3] and monoid of orbit graphs inspired by [19]. In the central Sect. 4 we state the thin-thick alternative for timed automata and a sort of pumping lemma for thick automata (Thm. 1). In Sect. 5 we apply these results to entropy of timed languages. We conclude with some perspectives in Sect. 6.

2 Preliminaries

2.1 Timed Languages and Their Measures

A *timed word* $\alpha = (t_1, a_1) \dots (t_n, a_n)$ is a word on the alphabet $\mathbb{R}^+ \times \Sigma$ where Σ is a finite alphabet of *events*. Times t_i represent *delays* between events a_{i-1} and a_i. Throughout this paper, delays will be bounded[1] by an integer constant M. We will sometimes write the same timed word α as $(\boldsymbol{t}, w)$ with $\boldsymbol{t} \in [0, M]^n$ and $w \in \Sigma^n$. A timed language L is a set of timed words. We will denote by L_n the language L restricted to words of length n.

Volume and volumetric entropy. Let L be a timed language. For each word of events $w \in \Sigma^n$, let $L(w)$ be the set $\{\boldsymbol{t} \in [0, M]^n \mid (\boldsymbol{t}, w) \in L\}$. This subset of $\mathbb{R}^n$ has a volume[2] (Lebesgue's measure) denoted by $\texttt{Vol}(L(w))$. The *volume* of L_n is $\texttt{Vol}(L_n) = \sum_{w \in \Sigma^n} \texttt{Vol}(L(w))$. Bounding the delays by M leads to a volume bounded by $(|\Sigma| M)^n$.

In this paper, we will work with factor closed languages[3], i.e. such that if $(t_1, a_1) \dots (t_n, a_n) \in L$ then for all $1 \leq i \leq j \leq n$ $(t_i, a_i) \dots (t_j, a_j) \in L$. For each couple of words $w_1, w_2 \in \Sigma^*$ the language inclusion $L(w_1 \cdot w_2) \subseteq L(w_1)L(w_2)$

[1] Our approach to timed languages is based on volumes, and does not apply, in its present form, to unbounded delays which lead to infinite volumes.

[2] Under the condition that the set is measurable; timed languages considered in this paper are all measurable as unions of polytopes.

[3] This roughly corresponds to automata where all states are both initial and final.

holds, and then $\mathtt{Vol}(L(w_1 \cdot w_2)) \leq \mathtt{Vol}(L(w_1))\mathtt{Vol}(L(w_2))$. The volumetric entropy of a timed language is a number in $[-\infty, +\infty)$ defined as

$$\mathcal{H}(L) = \lim_{n\to+\infty} \frac{1}{n} \log_2(\mathtt{Vol}(L_n)).$$

The limit (finite or $-\infty$) exists due to subadditivity of $\log_2 \mathtt{Vol}(L_n)$ wrt n and is upper bounded by $\log_2(|\Sigma|\, M)$. A timed language L is called *thin* if $\mathcal{H}(L) = -\infty$ ant *thick* otherwise.

To deal with finite objects, one can discretize all the previous continuous languages. Given an $\varepsilon = \frac{1}{N}$, $(N \in \mathbb{N})$ called *discretization step*, let us define $L_{\varepsilon,n} = L_n \cap (\varepsilon\mathbb{N} \times \Sigma)^n$, i.e. the set of words in L_n with all delays multiple of ε, $L_\varepsilon = \bigcup_{n\in\mathbb{N}} L_{\varepsilon,n}$, $L_\varepsilon(w) = L(w) \cap (\varepsilon\mathbb{N})^n$. The ε-*entropy* h_ε is defined as

$$h_\varepsilon(L) = \lim_{n\to\infty} \frac{1}{n} \log_2(|L_{\varepsilon,n}|).$$

We will relate the discrete entropy h_ε to the continuous one $\mathcal{H}$ in Sect. 5.

Open, closed and punctual languages. A language is said to be *open* if for all n, the set L_n is an open subset (for the product topology) of $\mathbb{R}^n \times \Sigma^n$. In other words, L_n should be a finite union of $O \times \{w\}$ where w is a word of events and O an open subset of $\mathbb{R}^n$. Interior and closure of L, denoted $\mathtt{int}(L)$ and $\overline{L}$ are defined in a natural way.

Taking closure or interior of a finite union of polytopes of $\mathbb{R}^n$ has no effect on its volume. Nevertheless the number of discrete points can drastically change if punctuality is allowed (see [5]). We call a language to be *punctual* if $\overline{\mathtt{int}(L)} \neq \overline{L}$.

2.2 Timed Automata and Their Languages

Clocks, zones and regions. Let X be a finite set of variables called *clocks*. Clocks have non negative values bounded by a constant M_c. A rectangular constraint is a formula of the form $x \sim c$ where $x \in X$, $c \in \mathbb{N}$, $\sim \in \{\leq, <, =, >, \geq\}$. A diagonal constraint is a formula of the form $x - y \sim c$ where $x, y \in X$. A guard is a finite conjunction of rectangular constraints. We denote by G the set of all guards. A *zone* is a set of clock vectors $\boldsymbol{x} \in [0, M_c]^X$ satisfying a finite conjunction of rectangular and diagonal constraints. A *region* is an inclusion-minimal zone. A region (which is always a simplex) is uniquely defined by a point with integer coordinates $\lfloor \boldsymbol{x} \rfloor \in \{0, \ldots, M_c\}^X$ giving integer part of clocks and an order on the fractional part of clocks $0 \sim_0 \{x_{i_1}\} \sim_1 \{x_{i_2}\} \sim_2 \cdots \sim_{|X|} 1$ where $\sim_0, \ \ldots, \ \sim_{|X|} \in \{<, =\}$. The closure of a region (abusively called closed region) can be obtained by replacing $<$ by $\leq$ and $>$ by $\geq$ in its definition.

Timed automata. A *timed automaton* is a tuple $(Q, X, \Sigma, E, \mathcal{L}, I, F)$ with Q a finite set of *locations*; X a set of bounded clocks; $E \subseteq Q \times G \times 2^X \times Q$ a finite set of edges; $\mathcal{L} : E \to \Sigma$ a labeling function on edges; $I \subseteq Q \times [0, M_c]^X$ the set of initial states; $F \subseteq Q \times [0, M_c]^X$ the set of final states.

By default all the *states* (elements of $Q \times [0, M_c]^X$) are initial and final, otherwise they are given by union of zones.

The clocks grow with the same (unit) speed and some of them are reset to 0 when passing through an edge. More formally, there is a *transition* $(q, \boldsymbol{x}) \xrightarrow{(t,a)} (q', \boldsymbol{x}')$ if there is an edge $e = (q, \mathfrak{g}, R, q') \in E$ with $\mathcal{L}(e) = a$ such that $\boldsymbol{x} + (t, \dots, t)$ satisfies the guard $\mathfrak{g}$ and for each clock $x \in X$, its new value $x' = 0$ iff $x \in R$, or $x' = x+t$ otherwise. A *run* on the timed word $\alpha = (t_1, a_1) \dots (t_n, a_n)$ is a sequence of consecutive transitions $(q_0, \boldsymbol{x}_0) \xrightarrow{(t_1,a_1)} (q_1, \boldsymbol{x}_1) \cdots \xrightarrow{(t_n,a_n)} (q_n, \boldsymbol{x}_n)$, where $\boldsymbol{x}_0, \dots, \boldsymbol{x}_n \in [0, M]^X$, $q_0, \dots, q_n \in Q$. A timed word is recognized by the automaton if there exists a run on it from an initial state to a final state. The timed language $L(\mathcal{A})$ consists of all the recognized words. We will be interested in its entropy $\mathcal{H}(L(\mathcal{A}))$, that will be abusively denoted $\mathcal{H}(\mathcal{A})$.

We call a TA *right resolving* if any two edges leaving the same location and having the same label have disjoint guards. Adding the condition that there is only one initial state gives the usual definition of determinism. In the rest of the paper, we work with right resolving TA.

Paths and reachability relation. We call a *path* in an automaton any sequence of edges. The "useful" paths are sequences of consecutive edges (such that the starting location of the $(i+1)^{th}$ edge is the ending one of the i^{th}), but we allow arbitrary words of E^* and all objects associated to a non-consecutive sequence will be empty.

Given two clock vectors $\boldsymbol{x}, \boldsymbol{x}'$, a path $\pi \in E^n$ and a sequence of delays of the same length $\boldsymbol{t} = (t_1, \dots, t_n)$, we write that $\boldsymbol{x} \xrightarrow{\boldsymbol{t},\pi} \boldsymbol{x}'$ whenever exists a run in the automaton of the form $(q_0, \boldsymbol{x}) \xrightarrow{(t_1,a_1)} (q_1, \boldsymbol{x}_1) \cdots \xrightarrow{(t_n,a_n)} (q_n, \boldsymbol{x}')$ following the sequence of edges π.

Several objects are naturally associated with a path. Given a path and two clock vectors, a language (a polytope of all the timings of the path) can be defined: $L(\pi, \boldsymbol{x}, \boldsymbol{x}') = \{\boldsymbol{t} \mid \boldsymbol{x} \xrightarrow{\boldsymbol{t},\pi} \boldsymbol{x}'\}$. If we are not interested in clock values, we get a polytope depending only on the path: $L(\pi) = \{\boldsymbol{t} \mid \exists \boldsymbol{x}, \boldsymbol{x}', \boldsymbol{x} \xrightarrow{\boldsymbol{t},\pi} \boldsymbol{x}'\}$. The other way around, if we do not care about timing, we get the reachability predicate: $\mathtt{Reach}(\pi) = \{(\boldsymbol{x}, \boldsymbol{x}') \mid \exists \boldsymbol{t}, \boldsymbol{x} \xrightarrow{\boldsymbol{t},\pi} \boldsymbol{x}'\}$.

A path $\pi \in E^n$ is said to be *punctual* if $L(\pi)$ is not empty but has dimension less than n. The closure (resp. interior) of a path π denoted by $\overline{\pi}$ (resp. $\mathtt{int}(\pi)$) is the path obtained from π by taking closure (resp. interior) of all guards of edges in π (i.e. changing strict inequalities in non-strict ones (resp. vice versa)). There is a *limit cycle* (resp. *strong limit cycle*) along π if there exists a clock vector $\boldsymbol{x}$ and time sequence $\boldsymbol{t}$ such that $\boldsymbol{x} \xrightarrow{\boldsymbol{t},\overline{\pi}} \boldsymbol{x}$ (resp. $\boldsymbol{x} \xrightarrow{\boldsymbol{t},\mathtt{int}(\pi)} \boldsymbol{x}$). Given $\varepsilon > 0$, in ε-*discrete limit cycles* all the components of $\boldsymbol{x}$ and $\boldsymbol{t}$ should be multiple of ε.

2.3 Thinness, Simplices and Examples

Our analysis of thin languages will start with a simple observation that the volume of k-dimensional simplices tends to 0 faster than any exponent:

Lemma 1. *The volume of a simplex of "type 1" described by inequalities* $0 \leq t_1 + \dots + t_k \leq 1$, $t_i \geq 0$ *or of a simplex of "type 2" described by inequalities* $0 \leq t_1 \leq \dots \leq t_k \leq 1$ *is* $\frac{1}{k!}$.

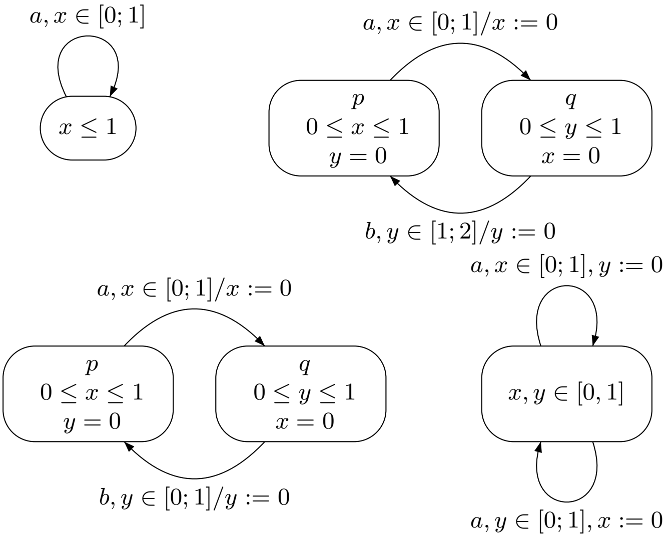


Fig. 1. First row: thin automata $\mathcal{A}_1$, $\mathcal{A}_2$. Second row: thick ones $\mathcal{A}_3$, $\mathcal{A}_4$. Initial states are given by conditions in nodes.

By change of coordinates the lemma can be extended to more general polytopes:

Corollary 1. *Let P be a subset of $\{t_1, \dots, t_n \mid 0 \le t_i \le M\}$. If there exists a subsequence of indices $s(1), \dots, s(k)$ of $1, \dots, n$ and new coordinates $u_{s(i)}$ functions of $t_{s(1)}, \dots, t_{s(i)}$ with $0 \le u_{s(1)} \le u_{s(2)} \le \dots \le u_{s(k)} \le 1$ and $\left|\frac{\partial u_{s(i)}}{\partial t_{s(i)}}\right| \ge 1$ then $\mathtt{Vol}(P) \le \frac{M^{n-k}}{k!}$.*

The automata on Fig. 1 illustrate the concepts of thin and thick. $L_n(\mathcal{A}_1) = \{t_1, \dots, t_n \mid \sum_{i \le n} t_i \le 1\}$ is a simplex of type 1, and thus $L(\mathcal{A}_1)$ is thin. $L_n(\mathcal{A}_2, q) = \{t_1, \dots, t_n \mid \forall i, t_{2i} + t_{2i+1} \le 1 \wedge t_{2i+1} + t_{2i+2} \ge 1\}$, by interchanging even and odd indices we obtain $L_n(\mathcal{A}_2, p)$. Posing $u_{2i+1} = 1 - t_{2i+1}$ and $u_{2i} = t_{2i}$ yields a simplex $0 \le u_1 \le \dots u_n \le 1$. This change of coordinates preserves volume and so $\mathtt{Vol}(L_n(\mathcal{A}_2, q)) = \frac{1}{n!}$. This is an example of automaton satisfying the *progress cycle condition* (i.e. resetting all clocks along each cycle) and nevertheless thin.

Third and fourth examples are thick, their entropies can be computed symbolically because they are $1\frac{1}{2}$ clock (see [3]), they are respectively $\log_2 \frac{2}{\pi}$ and $\log_2 \log_2(e)$. Note that $\mathcal{A}_4$ does not satisfy the progress cycle condition.

3 More on Paths and Cycles

3.1 Region Graph and State Split Automata

Timed variants of the region graph [2] are extensively used in the literature. Here we use so-called region-split automaton given in [3], add several new conditions

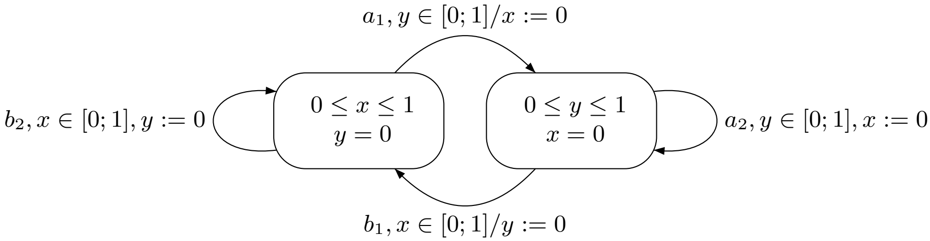

$a_1, y \in [0;1]/x := 0$
$b_2, x \in [0;1], y := 0$
$0 \leq x \leq 1$
$y = 0$
$0 \leq y \leq 1$
$x = 0$
$a_2, y \in [0;1], x := 0$
$b_1, x \in [0;1]/y := 0$

Fig. 2. The closed region-split version of $\mathcal{A}_4$

and modify those concerning initial states. A timed automaton is in region-split form if

B1. For every location $q \in Q$ a unique region $\mathbf{r}_q$ (called its *entry region*) exists, such that the set of clock values with which q is entered is exactly $\mathbf{r}_q$.
B2. The guard $\mathfrak{g}$ of every transition $\delta = (q, \mathfrak{g}, R, q') \in E$ is just one region.
B3. All the states of entry regions (and only these states) are both initial and final.
B4. For any location there exists a path leading to some cycle and a path coming from some cycle.
B5. For every transition δ its guard $\mathfrak{g}$ has no constraints of the form $x = c$ in its definition.
B6. The labeling function on edges is identity (and so every two distinct edges have different labels).

Proposition 1. *Given a right resolving TA $\mathcal{A}$ with bounded clocks and all states initial and final, one can construct an automaton $\mathtt{RS}(\mathcal{A})$ called the region-split automaton of $\mathcal{A}$ which satisfies $B1 - B6$ and such that $\mathcal{H}(\mathtt{RS}(\mathcal{A})) = \mathcal{H}(\mathcal{A})$.*

In the following, we replace w.l.o.g. for the computing of $\mathcal{H}$, $\mathcal{A}$ by $\overline{\mathtt{RS}(\mathcal{A})}$ obtained from $\mathtt{RS}(\mathcal{A})$ by taking non-strict inequalities instead of strict ones.

Proposition 2. *For region split automata:*

- *words, paths, and region paths are in natural bijection;*
- *volume of any path is less or equal to* 1*;*
- *every path of consecutive edges has a non empty and non punctual language.*

As for our running examples, $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ are already region split automata. $\overline{\mathtt{RS}(\mathcal{A}_4)}$ is depicted in Fig. 2.

3.2 Paths and Polytopes

Let us describe languages associated with paths as polytopes in $\mathbb{R}^n$, following [18].

Contiguous polytopes. Let $T_n = \{t_1, \ldots, t_n\}$ be an ordered set of real variables bounded by M. A sum $S_{j..k} = \sum_{i=j}^{k} t_i$ is called a *contiguous sum* (of *length* $k - j + 1$). A *temporal inequality* is a constraint of the form $S_{j..k} \in [A, B]$ where $A, B \in \mathbb{N}$, $A \geq 0$ and $B \leq M$. A *contiguous polytope* is a bounded subset of

$\mathbb{R}^n$ which is composed by all the points satisfying a conjunction of temporal inequalities. We say that the polytope is *d-contiguous* if the length of all sums in the inequalities is bounded by d.

Proposition 3. *For each path $\pi \in E^*$, $L(\overline{\pi})$ is a contiguous polytope.*

The inequality $\sum_{i=j}^{k} t_i \in [A, B]$ comes from testing the guard $x \in [A, B]$ during the k^{th} transition, provided that the last reset of x took place in the j^{th} transition. The subclass of d-contiguous polytopes corresponds to automata with progress cycle condition, where the number of transitions between two resets is bounded by d.

Given a polytope $\mathcal{P}$, we denote by $N\mathcal{P}$ its N-fold dilated copy, i.e. $\{N\boldsymbol{t} \mid \boldsymbol{t} \in \mathcal{P}\}$ and by $E(\mathcal{P}) = \mathcal{P} \cap \mathbb{Z}^n$ the set of points with integer coordinates in $\mathcal{P}$. A contiguous polytope is said to be *N-fat* if there exists an integer point in the interior of $N\mathcal{P}$ (called an *internal point*): $E(\mathtt{int}(N\mathcal{P})) \neq \emptyset$.

3.3 Point to Point Reachability: Algebraic Characterization

In this section, we characterize the relation $\mathtt{Reach}(\overline{\pi})$ in terms of an algebraic object: monoid of orbit graphs. Our analysis is less detailed than those in [10,11,16] and follows the lines of [19].

For a closed region $\overline{\mathbf{r}}$, let us denote by $V(\mathbf{r}) = \{S_1, \ldots, S_p\}$ its vertices. Any point $\boldsymbol{x}$ in the region is uniquely described by its barycentric coordinates $\lambda_1, \ldots, \lambda_p$, i.e. nonnegative numbers such that $\sum_{i=1}^{p} \lambda_i = 1; \quad \boldsymbol{x} = \sum_{i=1}^{p} \lambda_i S_i$.

Given two regions $\overline{\mathbf{r}}$ and $\overline{\mathbf{r}'}$, we call *orbit graph* any graph G with vertices $V(\mathbf{r}) \uplus V(\mathbf{r}')$ if $\mathbf{r}$ and $\mathbf{r}'$ are different and $V(\mathbf{r})$ otherwise, and with edges going from $V(\mathbf{r})$ to $V(\mathbf{r}')$. Informally, an edge from S to S' means that the clock vector at the vertex S can reach the clock vector at S' along some transition or path.

Orbit graphs compose in the natural way: for G_1 on regions $\overline{\mathbf{r}_1}$ and $\overline{\mathbf{r}'_1}$, and G_2 on regions $\overline{\mathbf{r}_2}$ and $\overline{\mathbf{r}'_2}$, their product $G = G_1 \cdot G_2$ is defined if $\overline{\mathbf{r}'_1} = \overline{\mathbf{r}_2}$. In this case, G is an orbit graph on $\overline{\mathbf{r}_1}$ and $\overline{\mathbf{r}'_2}$. There is an edge from S to S'' in G if and only if there exists S' such that (S, S') and (S', S'') are edges of G_1 and G_2. Whenever $\overline{\mathbf{r}'_1} \neq \overline{\mathbf{r}_2}$, we put $G_1 \cdot G_2$ equal to some special (absorbing) element $\mathbf{0}$. The set $\mathcal{G}$ of orbit graphs, augmented with $\mathbf{0}$ and a neutral element $\mathbf{1}$ has a structure of finite monoid.

An orbit graph G can be represented by its adjacency matrix M of size $|V(\mathbf{r})| \times |V(\mathbf{r}')|$. Products in the monoid of orbit graphs are easy to compute using matrices: $M(G_1G_2) = M(G_1) \otimes M(G_2)$ where the "product" $\otimes$ is defined by

$$(A \otimes B)_{ij} = \max_k \min(A_{ik}, B_{kj}).$$

There exists a natural morphism $\gamma : E^* \to \mathcal{G}$ from paths to orbit graphs defined as follows. For a transition e between $\overline{\mathbf{r}}$ and $\overline{\mathbf{r}'}$, we define the orbit graph $\gamma(e)$ on $\mathbf{r}$ and $\mathbf{r}'$ with edges $\{(S, S') \in V(\mathbf{r}) \times V(\mathbf{r}') \mid \exists t, S \xrightarrow{(e,t)} S'\}$. For a path $\pi = e_1 \ldots e_n$, we define $\gamma(\pi) = \gamma(e_1) \ldots \gamma(e_n)$ (it will be called the orbit graph of the path π). For the empty path we have $\gamma(\varepsilon) = \mathbf{1}$, and for any non-consecutive path $\gamma(\pi) = \mathbf{0}$.

For example, the orbit graphs of cycles ab and ba of $\mathcal{A}_3$ and $\mathcal{A}_4$ are complete, the orbit graphs of the other running examples are given in Fig. 3.

The orbit graph is crucial for reachability analysis.

Proposition 4. *The orbit graph of a path $\gamma(\pi)$ determines its reachability relation $\mathtt{Reach}(\bar{\pi})$. In particular, $\gamma(\pi)$ is complete iff $\mathtt{Reach}(\bar{\pi}) = \overline{\mathbf{r}} \times \overline{\mathbf{r}'}$, or equivalently iff $\mathtt{Reach}(\mathtt{int}(\pi)) = \mathbf{r} \times \mathbf{r}'$.*

The proof of the first criterion is based on the following remarkable characterization of $\mathtt{Reach}(\bar{\pi})$ in terms of the orbit graph due to Puri [19].

Lemma 2. [4] *Let $\boldsymbol{x}$ and $\boldsymbol{x}'$ be two clock vectors with barycentric coordinates $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}'$. Then $(\boldsymbol{x}, \boldsymbol{x}') \in \mathtt{Reach}(\bar{\pi})$ iff there exists a stochastic matrix $P \preceq M(\gamma(\pi))$, such that $\boldsymbol{\lambda}P = \boldsymbol{\lambda}'$.*

Here matrix "inequality" $A \preceq B$ means that $B_{ij} = 0 \Rightarrow A_{ij} = 0$ for all i, j.

Adding clock resets. For future use, we must enrich the monoid of orbit graphs by adding information on clock resets. Elements of the monoid $\mathcal{M}$ are couples (orbit graph, subset of clocks) (and also, as before, two special elements $\mathbf{0}, \mathbf{1}$), the product rule is:

$$(G_1, X) \cdot (G_2, Y) = \begin{cases} (G_1 \cdot G_2, X \cap Y), & \text{if } G_1 \cdot G_2 \neq \mathbf{0} \\ \mathbf{0}, & \text{otherwise.} \end{cases}$$

For each $\pi \in E^*$ we denote by $\nu(\pi)$ the set of clocks not reset along the path π. We define a morphism $\mu : E^* \to \mathcal{M}$ as follows: $\mu(\pi) = (\gamma(\pi), \nu(\pi))$.

Idempotents. An idempotent of a monoid is an element m such that $m \cdot m = m$. Every finite monoid contains an idempotent. In our case, an idempotent orbit graph is always associated to a cyclic path, it is a graph G equal to its transitive closure $G^+ = \cup_{n \in \mathbb{N}^+} G^n$.

4 The Thin-Thick Alternative and Its Consequences

In this central section, we characterize thin and thick paths and languages, based on a new notion of a forgetful cycle.

4.1 Forgetful Cycles, and the Others

After reading a timed path $\pi \times \boldsymbol{t}$ from a state s_0, the reached state s depends only on s_0 and on the delays $\boldsymbol{t}$. We will say that π is *forgetful* if s and s_0 are independent, i.e. all the following equivalent conditions hold: $\mathtt{Reach}(\mathtt{int}(\pi)) = \mathbf{r} \times \mathbf{r}'$, $\mathtt{Reach}(\bar{\pi}) = \bar{\mathbf{r}} \times \bar{\mathbf{r}}'$, $\gamma(\pi)$ is complete.

If a cycle is non-forgetful, and moreover its orbit graph is not strongly connected, then it is possible to find a linear Lyapunov function:

[4] An intuition behind this lemma could be as follows. A clock vector with barycentric coordinates $\boldsymbol{\lambda}$ in a region can be seen as a probabilistic distribution over vertices of this region (with probabilities $\boldsymbol{\lambda}$). The lemma says that this distribution, at each cycle, evolves exactly as in some Markov chain.

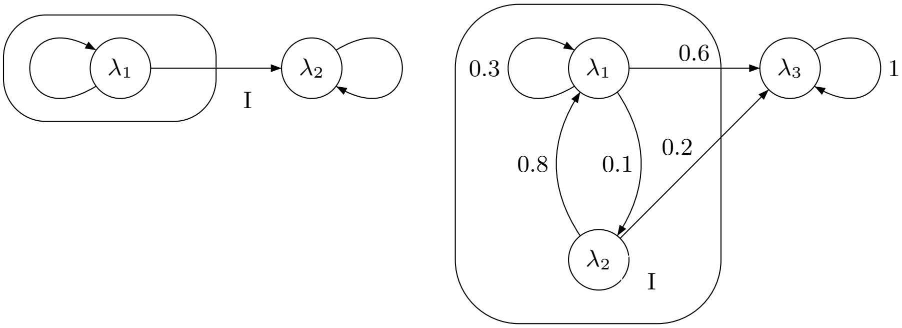


Fig. 3. Two non strongly connected orbit graphs, the first one is the orbit graph of the cycle of $\mathcal{A}_1$, of the cycle ab of $\mathcal{A}_2$ and of the cycles a and b of $\mathcal{A}_4$. States move from initial SCC I to final one. By choosing the convex combination of paths given by the Markov chain on the second orbit graph we pass from state $(\lambda_1 = 0.2, \lambda_2 = 0.5, \lambda_3 = 0.3)$ to state $(\lambda_1' = 0.46, \lambda_2' = 0.02, \lambda_3' = 0.52)$. The sum $\lambda_1 + \lambda_2$ can only decrease.

Lemma 3. *For a cycle π, if $\gamma(\pi)$ is not strongly connected then there exists a non empty $I \subsetneq \{1, \ldots, p\}$ such that $\sum_{i \in I} \lambda_i' \leq \sum_{i \in I} \lambda_i$ whenever $(\boldsymbol{x}, \boldsymbol{x}') \in$ $\texttt{Reach}(\bar{\pi})$, where $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}'$ stand for barycentric coordinates of $\boldsymbol{x}$ and $\boldsymbol{x}'$.*

In this lemma, as before, $\{1, \ldots, p\}$ are indices of the vertices of the region where π starts (and ends).

In fact I corresponds to an initial strongly connected component (SCC) of the orbit graph, i.e. an SCC without incoming edges from other SCC. According to the lemma, the state moves from the facet spanned by I towards other vertices of the region and cannot come back.

Comparing to other types of cycles. Two other kinds of cycles are often considered in the literature: in a *progress cycle* each clock should be reset at some edge; in a *synchronizing cycle* all the clocks are reset along one and the same edge of the cycle.

Proposition 5. *progress cycles $\supsetneq$ forgetful cycles $\supsetneq$ synchronizing cycles.*

A remark is in order, in most works using progress or synchronizing cycles, **all the cycles** are **required** to be like that. In our work, **existence** of a forgetful cycle **appears** naturally in "non degenerate" (i.e. thick) automaton.

The condition of progress cycle can be seen as a weaker kind of forgetting: the state after such a cycle is exactly determined by the delays (see following lemma). Nevertheless the orbit graph of a progress cycle can be not strongly connected (e.g. cycle ab of $\mathcal{A}_2$ depicted in Fig. 3); in that case starting states and ending states are still dependent.

Lemma 4. *If all clocks have been reset during reading of $\pi \times (t_1, \ldots, t_m)$ then for all non empty $I \subsetneq \{1, \ldots, p\}$, there exists $\alpha_1 \ldots \alpha_m \in \{-p, \ldots, p\}$ and an integer constant C such that $\sum_{i \in I} \lambda_i = C + \sum_{j=1}^{m} \alpha_j t_j$. Moreover one of the α_j is not zero.*

4.2 Pumping Lemma for Long Thick Paths

For a given real $\eta > 0$, we say that a path π is *η-thick* if $\mathtt{Vol}(L(\pi)) \geq \eta^{|\pi|}$. The following "pumping lemma" will play the key role in characterization of thick languages below and can be interesting by itself.

Theorem 1. *For every timed automaton $\mathcal{A}$ and every $\eta > 0$, there exists N_η such that any η-thick path longer than N_η contains a forgetful cycle.*

The rest of this section is devoted to the proof of this result.

Elements of the monoid $\mathcal{M}$ associated to forgetful cycles will be referred to as *forgetful*, they are idempotent. We will first see how repeating a non forgetful idempotent induces a subexponential volume (like the simplex example), then we will use Simon's theorem on factorization forests to factorize paths and find some repeated idempotent. Absence of forgetful cycles in a path will then imply thinness.

Proposition 6. *Let $\pi_1, \ldots, \pi_k$ be k cycles of E^* such that $\mu(\pi_1), \ldots, \mu(\pi_k)$ are all equal to a same non forgetful idempotent of $\mathcal{M}$, then $\mathtt{Vol}(L(\pi_1 \ldots \pi_k)) \leq \frac{M^{n-k}}{k!}$ where $n = |\pi_1| + \cdots + |\pi_k|$.*

Proof. If G is an idempotent orbit graph (thus equal to its transitive closure), G is complete if and only if G is strongly connected. Thus we will distinguish two disjoint kinds of non forgetful idempotents, those associated to non progress cycles and those associated to progress cycles with non strongly connected orbit graphs. In the former case a clock is not reset all along the path $\pi_1 \ldots \pi_k$, thus $L(\pi_1 \ldots \pi_k)$ is in a simplex of type 1 and the volume satisfies the inequality to prove. In the latter case, we use Lem. 3,4, and Cor. 1. □

A *factorization forest* of a word π is an unranked labeled tree with leaves labeled by the letters of π, with root labeled by π and with two types of internal nodes:

– binary node labeled by a word $\pi_1 \cdot \pi_2$ with two children labeled by the words π_1 and π_2;
– idempotent node labeled by a word $\pi_1 \ldots \pi_k$ with all $\mu(\pi_i)$ equal to a same idempotent and with children labeled by the words $\pi_1, \ldots, \pi_k$.

Theorem (Simon [20]). *If μ is a morphism from E^* to a finite monoid $\mathcal{M}$, then every word admits a factorization forest of height at most $h(\mathcal{M}) = 9|\mathcal{M}|$.*

We suppose that there are no forgetful cycles on a long path π and consider its factorization forest of height at most $h(\mathcal{M})$. When its length n grows up, the number of leaves also grows and since the height is bounded, branching of nodes must get larger and larger. These hugely branched nodes are idempotent and satisfy hypotheses of Lem. 6, thus their volume is very small, which implies that $\mathtt{Vol}(L(\pi))$ is also small. The Prop. 7 below quantifies this "smallness" of $\mathtt{Vol}(L(\pi))$ as function of the length of π and height of its factorization forest, and Thm. 1 follows immediately from this proposition.

Let $\mathtt{LVol}$ be the function defined on paths by $\mathtt{LVol}(\pi) = \log_2 \mathtt{Vol}(L(\pi))$. This function is subadditive and non-positive, i.e. $\mathtt{LVol}(\pi_1 \cdot \pi_2) \le \mathtt{LVol}(\pi_1) + \mathtt{LVol}(\pi_2) \le 0$. Let $L(n,h)$ be the maximum of $\mathtt{LVol}(\pi)$ over paths π of length n that do not contain forgetful idempotents and admit a factorization forest of height at most h.

Proposition 7. *For any height h, for any $C < 0$, there exists $N_{h,C} \in \mathbb{N}$ such that for all $n > N_{h,C}$ the inequality $L(n,h) \le Cn$ holds.*

Proof. We will define $N_{h,C}$ by induction on the height h. Let a be a factorization forest of height h with n leaves. We consider all the children of the root and their subtrees (all these subtrees have heights $\le h-1$), and distinguish two disjoint cases:

1. There are more than $k = \frac{n}{2N_{h-1,2C}}$ subtrees having less than $N_{h-1,2C}$ leaves.
2. There are less than $k = \frac{n}{2N_{h-1,2C}}$ subtrees with less than $N_{h-1,2C}$ leaves. Here the juicy part (sons with enough leaves to satisfy induction hypothesis) has more than $\frac{n}{2}$ leaves.

In the first case: root is an idempotent node and we can apply Lem. 6:

$$\mathtt{LVol}(\pi) \le (n-k)\log_2(M) - \log_2(k!) \le nC \text{ for } n \text{ large enough.}$$

In the second case: $\mathtt{LVol}(\pi) \le \sum_{i=1}^{k} L(n_i,h_i) \le \sum_{n_i \ge N_{h-1,2C}} L(n_i,h_i)$. We apply the induction hypothesis:

$$\mathtt{LVol}(\pi) \le 2C \sum_{n_i \ge N_{h-1,2C}} n_i \le 2C\frac{n}{2} \le nC \text{ (recall that } C \text{ is negative).} \qquad \square$$

To conclude the proof of Thm. 1, given $\eta > 0$, let $C = \log_2 \eta$ and $h = h(\mathcal{M})$ the bound on height of factorization forest. Using Prop. 7, we obtain that a path longer than $N_{h,C}$ without forgetful idempotents cannot be η-thick. □

4.3 Characterizing Thick Languages

We are ready to describe thick languages now.

Theorem 2. *For a right resolving timed automaton in region split form the following conditions are equivalent and define thick languages:*

1. *$\mathcal{H} > -\infty$;*
2. *there exists a forgetful cycle;*
3. *there exists a strong limit cycle;*
4. *there exists an ε-discrete strong limit cycle with $\varepsilon > 0$.*

Equivalence between 3 and 4 can be found in [16]. 2 ⇒ 3 is straightforward.

Proof of 4 ⇒ 1. There exist $q_0, \dots, q_{d-1}, \boldsymbol{x}_0, \dots, \boldsymbol{x}_{d-1}$, $\pi \in E^d, u_1, \dots, u_d \in \{\varepsilon, 2\varepsilon, \dots, M - \varepsilon\}$ such that $(q_0, \boldsymbol{x}_0) \xrightarrow{(u_1,w_1)} (q_1, \boldsymbol{x}_1) \dots \xrightarrow{(u_d,w_d)} (q_0, \boldsymbol{x}_0)$ along π and all the $\boldsymbol{x}_i$ are not on the frontier of regions and have discrete coordinates. First we can see that all clocks have been reset at least once because any non-reset clock would augment during the run, which contradicts its cyclicity. Then for each $n \in \mathbb{N}^*$ the language $L(\pi^n)$ is a d-contiguous polytope with equation of the form $A \leq \sum_{i=j}^{k} t_i \leq B$. Extending u periodically permits to have a word in $L(\pi^n)$ such that $A + \varepsilon \leq \sum_{i=j}^{k} u_i \leq B - \varepsilon$. Taking $t_i \in [u_i - \frac{\varepsilon}{d}, u_i + \frac{\varepsilon}{d}]$ defines a hypercube included in $L(\pi^n)$ whose volume is therefore greater than $(\frac{2\varepsilon}{d})^{nd}$. Then $\mathcal{H}(\mathcal{A}) \geq \log_2 \frac{2\varepsilon}{d} > -\infty$. □

Proof of 1 ⇒ 2. We notice first that a thick language contains long thick paths.

Lemma 5. *If $\mathcal{H} > -\infty$, there exists $\eta > 0$ such that for all n big enough, there exists an η-thick path of length n.*

Proof. Let $\beta = 2^{\mathcal{H}-1}$. For n large enough $\texttt{Vol}(L_n) \geq \beta^n$. Let $\pi_{n,\max}$ be the path of E^n of maximal volume, then $\texttt{Vol}(L_n) \leq \texttt{Vol}(L(\pi_{n,\max}))|E|^n$ and so if we pose $\eta = \frac{\beta}{|E|}$ we have $\texttt{Vol}(L(\pi_{n,\max})) \geq \eta^n$. □

Combining Lem. 5 with Thm. 1 we find a required forgetful cycle. □

4.4 Thin and Thick SCC

The theory developed above can be refined using a decomposition of $\mathcal{A}$ into strongly connected components (SCC) $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_k$.

Proposition 8. *Volumetric entropy of $\mathcal{A}$ equals the maximal volumetric entropy of its SCC. In particular, $\mathcal{A}$ is thin iff so are all the subautomata $\mathcal{A}_i$.*

It is easy to see that long and thick paths spend most of the time in thick SCC.

Theorem 3. *For every timed automaton $\mathcal{A}$ and every $\eta, \alpha > 0$, there exists $N_{\eta,\alpha}$ such that for any η-thick path of length $n > N_{\eta,\alpha}$ at most $n\alpha$ states belong to thin SCC.*

5 Entropies of Thick Languages

In this section, we apply the results of the previous section to show that in thick automata, volumes and entropies can be computed with a good precision using discretization.

Theorem 4. *For a thick strongly connected automaton $\mathcal{A}$ in region split form, the discrete and the volumetric entropies are related as follows*[5]*:*

$$h_\varepsilon = \log_2 \frac{1}{\varepsilon} + \mathcal{H} + o(1).$$

[5] It can be proved that $o(1)$ is in fact $O\left(\varepsilon^{1/3}\left(\log_2 \frac{1}{\varepsilon}\right)^{2/3}\right)$.

Proof ($\geq$ direction). To bound the volume of L_n by the number of discrete points, we will use a beautiful theorem on counting points in polytopes:

Theorem (Ehrhart, see [8]). *For integer N and an integer polytope $\mathcal{P} \subset \mathbb{R}^n$ (i.e. whose vertices have integer coordinates), the number of integer points $|E(N\mathcal{P})|$ is a polynomial in N with non negative coefficients of degree n and whose coefficient of the highest degree is the volume.*

We deduce directly from this theorem that for each path π of length n and $\varepsilon = \frac{1}{N}$ the following holds: $\texttt{Vol}(L(\pi))N^n \leq |E(NL(\pi))| = |L_\varepsilon(\pi)|$. Summing over all words of length n and taking $\lim_{n\to\infty} \frac{1}{n}\log_2$, we obtain that $\mathcal{H}+\log_2\frac{1}{\varepsilon} \leq h_\varepsilon$. □

Proof ($\leq$ direction of Thm. 4).

Upper bounding h_ε by $\mathcal{H} + \log_2\frac{1}{\varepsilon} + o(1)$ is more involved, and we give only a sketch of proof. We fix several integer parameters: a, b, c, d, e (they have to be adjusted in order to obtain the required estimate). Let π be a path of a length $n > a$. At every b transitions, we insert in π a forgetful cycle of length c (it exists by virtue of Thm. 2, and can be made of the same fixed length everywhere for an appropriate choice of c). Thus we obtain a slightly longer path π' (its length is $n' \approx n(1 + c/b)$), satisfying two additional conditions:

- every clock is reset at least every $2d$ transitions (and thus $L(\pi')$ is $2d$-contiguous polytope);
- the polytope $L(\pi')$ is e-fat.

We have three inequalities:

1. The first one:
$$|L_\varepsilon(\pi)| \leq |L_\varepsilon(\pi')|$$
can be proved by constructing an injection from the left-hand side discrete language to the right-hand side one.
2. We choose ε' slightly smaller than ε (another parameter to adjust) and consider the polytope L^- obtained from $L(\pi')$ by pushing all its facets inside by the amount $\delta = \varepsilon' d$ [6]. Using fatness of $L(\pi')$, it is possible to build an injection from its ε-discrete points to ε'-discrete points of L^- (the latter is a bit smaller but its discrete points are slightly denser).
$$|L_\varepsilon(\pi')| \leq \left|L^-_{\varepsilon'}\right|.$$
3. Taking an ε'-cube at every ε'-discrete point of L^-, we get a set included in $L(\pi')$ (this requires $2d$-contiguity of $L(\pi')$). Passing to volumes we conclude that
$$\varepsilon'^{n'}\left|L^-_{\varepsilon'}\right| \leq \texttt{Vol}(L(\pi')).$$

[6] i.e. by replacing each constraint $S_{j..k} \in [A, B]$ in the definition of $L(\pi')$ as a contiguous polytope by $S_{j..k} \in [A + \delta, B - \delta]$ (see [4]).

Combining the three inequalities we get:

$$|L_\varepsilon(\pi)| \le \left(\varepsilon'^{-n'}\mathtt{Vol}(L(\pi'))\right),$$

and with an appropriate choice of parameters, ε' and n' can be made very close to ε and n. Summing up over π and taking $\lim_{n\to\infty}\frac{1}{n}\log_2$ in the previous inequality, we obtain the required result. □

Corollary 2. *For right resolving TA with bounded clocks, $\mathcal{H}(\mathcal{A})$ is computable as function of $\mathcal{A}$. Consequently, $\mathcal{H}(\mathcal{A})$ is a computable real (i.e. one can compute its approximation with any wanted precision).*

Proof. First compute $\mathtt{RS}(\mathcal{A})$. Then compute by fixpoint method the submonoid of orbit graphs $\gamma(E^*) \subset \mathcal{G}$ and see whether there is a complete graph. If there is none, the automaton is thin and $\mathcal{H} = -\infty$. Otherwise, the automaton is thick and it just remains to compute the discrete entropy of $L_\varepsilon(\mathtt{RS}(\mathcal{A}))$ for the wanted precision (similarly to [4]). □

6 Conclusion and Future Work

We have identified the class of thick timed automata (those with non-vanishing language volume). Most runs in such automata are thick and exhibit a nice behavior, they spend most of the time in thick strongly connected components (Thm. 3) and visit from time to time forgetful cycles (Thm. 1). Thick runs are captured (both qualitatively and quantitatively) by ε-discretized automata.

We believe that the notions of thick languages and forgetful cycles will be useful in the operator approach to volume and entropy of [3] and will imply some good properties of operators associated to these forgetful cycle. Similarly, we believe that thickness hypothesis is exactly what is needed for the analysis of probabilistic timed systems in the spirit of [9] but for an unbounded time horizon. Another direction of future work is to extend the thin-thick dichotomy to the case of punctual paths and to find when the two size measures of [5] are defined. We hope also to relate thinness with the notion of mean topological dimension [17]. In the verification context, we believe that when analyzing a thick timed automaton, it suffices to check that the thick paths satisfy the specification, while thin ones can violate it.

Acknowledgements. The authors are thankful to Thomas Colcombet for a key advise: to use Simon factorization forests, and to Dominique Perrin and Aldric Degorre for inspiring discussions.

References

1. Abdulla, P.A., Krcál, P., Yi, W.: Sampled semantics of timed automata. Logical Methods in Computer Science 6(3) (2010)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126, 183–235 (1994)

3. Asarin, E., Degorre, A.: Volume and entropy of regular timed languages: Analytic approach. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 13–27. Springer, Heidelberg (2009)
4. Asarin, E., Degorre, A.: Volume and entropy of regular timed languages: Discretization approach. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 69–83. Springer, Heidelberg (2009)
5. Asarin, E., Degorre, A.: Two size measures for timed languages. In: FSTTCS, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. LIPIcs, vol. 8, pp. 376–387 (2010)
6. Asarin, E., Maler, O., Pnueli, A.: On discretization of delays in timed automata and digital circuits. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 470–484. Springer, Heidelberg (1998)
7. Beauquier, D.: Pumping lemmas for timed automata. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, pp. 81–94. Springer, Heidelberg (1998)
8. Beck, M., Robins, S.: Computing the continuous discretely: Integer-point enumeration in polyhedra. Springer, Heidelberg (2007)
9. Carnevali, L., Grassi, L., Vicario, E.: State-density functions over DBM domains in the analysis of non-Markovian models. IEEE Trans. Software Eng. 35(2), 178–194 (2009)
10. Comon, H., Jurski, Y.: Timed automata and the theory of real numbers. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 242–257. Springer, Heidelberg (1999)
11. Dima, C.: Computing reachability relations in timed automata. In: LICS, p. 177. IEEE Computer Society, Los Alamitos (2002)
12. Gómez, R., Bowman, H.: Efficient detection of Zeno runs in timed automata. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 195–210. Springer, Heidelberg (2007)
13. Gupta, V., Henzinger, T.A., Jagadeesan, R.: Robust timed automata. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 331–345. Springer, Heidelberg (1997)
14. Henzinger, T.A., Manna, Z., Pnueli, A.: What good are digital clocks? In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 545–558. Springer, Heidelberg (1992)
15. Henzinger, T.A., Raskin, J.F.: Robust undecidability of timed and hybrid systems. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 145–159. Springer, Heidelberg (2000)
16. Krcál, P.: Infinite Structures in Timed Systems. Ph.D. thesis, University of Uppsala, Dept. of Information Technology (May 2009)
17. Lindenstrauss, E., Weiss, B.: Mean topological dimension. Israel J. Math 115, 1–24 (2000)
18. Maler, O., Pnueli, A.: On recognizable timed languages. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 348–362. Springer, Heidelberg (2004)
19. Puri, A.: Dynamical properties of timed automata. Discrete Event Dynamic Systems 10(1-2), 87–113 (2000)
20. Simon, I.: Factorization forests of finite height. Theor. Comput. Sci. 72(1), 65–94 (1990)
21. Wulf, M.D., Doyen, L., Markey, N., Raskin, J.F.: Robust safety of timed automata. Formal Methods in System Design 33(1-3), 45–84 (2008)
22. Wulf, M.D., Doyen, L., Raskin, J.F.: Almost ASAP semantics: from timed models to timed implementations. Formal Asp. Comput. 17(3), 319–341 (2005)

Robust Specification of Real Time Components

Kim G. Larsen[1], Axel Legay[2], Louis-Marie Traonouez[3], and Andrzej Wąsowski[3]

[1] Aalborg University, Denmark
kgl@cs.aau.dk
[2] INRIA Rennes, France
axel.legay@irisa.fr
[3] IT University of Copenhagen, Denmark
{lmtr,wasowski}@itu.dk

Abstract. Specification theories for real-time systems allow to reason about interfaces and their implementation models, using a set of operators that includes satisfaction, refinement, logical and parallel composition. To make such theories applicable throughout the entire design process from an abstract specification to an implementation, we need to be able to reason about possibility to effectively implement the theoretical specifications on physical systems. In the literature, this implementation problem has already been linked to the robustness problem for Timed Automata, where small perturbations in the timings of the models are introduced. We address the problem of robust implementations in timed specification theories. Our contributions include the analysis of robust timed games and the study of robustness with respect to the operators of the theory.

1 Introduction

For long, software engineers have practiced component-oriented software construction, building systems from modules that only depend on each other in well specified ways. Foundational research follows up by developing trustworthy rigorous methods for component-oriented design. In concurrency theory this includes compositional design (specification theories, stepwise-refinement) and compositional model checking. Akin to algebraic specifications, specification theories provide a language for specifying component interfaces together with operators for combining them, such as parallel (structural) composition and conjunction (logical composition).

Specification theories integrate prior results to provide a uniform design method. In [10], we have proposed the first complete specification theory for timed systems. We build on an input/output extension of the classical timed automata model—inputs are used to represent behaviors of the environment and outputs represent the behavior of the component. The theory is equipped with a game-based semantic, which is used to define all the good operations for such specification theory, including satisfaction (can a specification be implemented), refinement (how to compare specifications), logical composition (computing the intersection of two sets of implementations), structural composition (building large components from smaller ones), and quotient (synthesizing a component in a large design).

The theory in [10] is equipped with a consistency check that allows to decide whether a specification can indeed be implemented. Unfortunately, this check does not take

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 129–144, 2011.

imprecision of the physical world into account, that is consistency can be used to synthesize an implementation that may be not robust with respect to variations of the environment. In practice, one would want to guarantee that a perturbation of the implementation still matches the requirements of the specification. Providing a solution to this problem in the setting of timed I/O specifications is our objective in this paper.

Our contributions include:

- We propose a notion of implementation of a specification that is robust with respect to a given perturbation in the delay before an action. Such perturbation is fixed in advance, which is a reasonable assumption as sensitivity with respect to perturbation of the environment is generally given in the description of the component that is provided by the manufacturer. The concept of robust implementation is lifted to a robust satisfaction operation that takes variations of timed behaviors into account when checking whether the implementation matches the requirement of the specification.
- We propose a consistency check for robust satisfaction. This new check relies on an extension of the classical timed I/O game to the robust setting. In [9], Chatterjee et al. showed that problems on robust timed games can be reduced to classical problems on an extended timed game. We modify the original construction of [9] to take the duality of inputs and outputs into account. Then, we show how our new game can be used to decide consistency in a robust setting as well as to synthesize a robust implementation from a given specification.
- Finally, classical compositional design operators are lifted to the robust setting. One of the nice features of our approach is that this does not require to modify the definitions of the operators themselves and that all the good properties of a specification theory (including independent implementability) are maintained.

To the best of our knowledge, this paper presents the first complete theory for robust timed specification. While the presentation is restricted to the theory of [10], we believe that the approach works for any timed specifications. Our experience with industrial projects shows that such realistic design theories are of clear interest [15,16,4].

State of The Art. None of the existing specification theories for timed systems allows for the treatment of robustness.

Various works have considered robustness for timed automata using logical formulas as specifications (and neglecting compositional design operators). The robust semantics for timed automata with clock drifts has been introduced by Puri [14]. The problem has been linked to the implementation problem in [18], which introduced the first semantics that modeled the hardware on which the automaton is executed. In this work, the authors proposed a robust semantics of Timed Automata called AASAP semantics (for "Almost As Soon As Possible"), that enlarges the guards of an automaton by a delay Δ. This work has been extended in [17] that proposes another robust semantics with both clock drifts and guard enlargement. Extending [14] they solve the robust safety problem, defined as the existence of a non-null value for the delays. They show that in terms of robust safety the semantics with clock drifts is just as expressive as the semantics with delay perturbation. We extend the work of [18,17] by considering compositional design operators, stepwise-refinement, and reasoning about open systems (only closed system composition were considered so far).

We solve games for consistency and compatibility using a robust controller synthesis approach largely inspired by Chatterjee et al. [9], who provide synthesis techniques for robust strategies in games with parity objectives. Driven by the fact that consistency and compatibility are safety games, we restrict ourselves to safety objectives, but we extend [9] by allowing negative perturbation of delays.

We proceed by introducing the background on Timed Specifications (Section 2). In Section 3 we introduce methods for solving robust time games that arise in our specification theory. These methods are used in Sections 4–5 to reason about consistency, conjunction, parallel composition and synthesis of specifications and robust implementations of real time components.

2 Background on Timed I/O Specifications

We now recall the definition of Timed I/O specifications [10]. We use $\mathbb{Z}$ (respectively $\mathbb{N}$) for the set of all integer numbers (resp. non-negative integers), $\mathbb{R}$ for the set of all real numbers, and $\mathbb{R}_{\geq 0}$ (resp. $\mathbb{R}_{>0}$) for the non-negative (resp. strictly positive) subset of $\mathbb{R}$. Rational numbers are denoted by $\mathbb{Q}$, and their subsets are denoted analogously. For $x \in \mathbb{R}_{\geq 0}$, let $\lfloor x \rfloor$ denote the integer part of x and $\langle x \rangle$ denote its fractional part.

In the framework of [10], specifications and their implementations are semantically represented by Timed I/O Transition Systems (TIOTS) that are nothing more than timed transition systems with input and output modalities on transitions. Later we shall see that input represents the behaviors of the environment in which a specification is used, while output represents behaviours of the component itself.

Definition 1. *A* Timed I/O Transition System *is a tuple $S = (St^S, s_0, \Sigma^S, \rightarrow^S)$, where St^S is an infinite set of states, $s_0 \in St^S$ is the initial state, $\Sigma^S = \Sigma_i^S \oplus \Sigma_o^S$ is a finite set of actions partitioned into inputs Σ_i^S and outputs Σ_o^S, and $\rightarrow^S : St^S \times (\Sigma^S \cup \mathbb{R}_{\geq 0}) \times St^S$ is a transition relation. We write $s \xrightarrow{a}^S s'$ when $(s, a, s') \in \rightarrow^S$ and use $i?$, $o!$ and d to range over inputs, outputs and $\mathbb{R}_{\geq 0}$, respectively.*

In what follows, we assume that any TIOTS satisfies the following conditions:

- time determinism: whenever $s \xrightarrow{d}^S s'$ and $s \xrightarrow{d}^S s''$ then $s' = s''$
- time reflexivity: $s \xrightarrow{0}^S s$ for all $s \in St^S$
- time additivity: for all $s, s'' \in St^S$ and all $d_1, d_2 \in \mathbb{R}_{\geq 0}$ we have $s \xrightarrow{d_1+d_2}^S s''$ iff $s \xrightarrow{d_1}^S s'$ and $s' \xrightarrow{d_2}^S s''$ for an $s' \in St^S$

A *run* ρ of a TIOTS S from its state s_1 is a sequence $s_1 \xrightarrow{a_1}^S s_2 \xrightarrow{a_2}^S \ldots \xrightarrow{a_n}^S s_{n+1}$ such that for all $1 \leq i \leq n$ $s_i \xrightarrow{a_i}^S s_{i+1}$. We write $\mathsf{Runs}(s_1, S)$ for the set of runs of S starting in s_1 and $\mathsf{Runs}(S)$ for $\mathsf{Runs}(s_0, S)$. We write $\mathsf{States}(\rho)$ for the set of states reached in ρ, and if ρ is finite $\mathsf{last}(\rho)$ is the last state occurring in ρ.

A TIOTS S is *deterministic* iff $\forall a \in \Sigma^S \cup \mathbb{R}_{\geq 0}$, whenever $s \xrightarrow{a}^S s'$ and $s \xrightarrow{a}^S s''$, then $s' = s''$. It is *input-enabled* iff each of its states $s \in St^S$ is input-enabled: $\forall i? \in \Sigma_i^S.\ \exists s' \in St^S.\ s \xrightarrow{i?}^S s'$. We say that S is *output urgent* iff $\forall s, s', s'' \in St^S$ if $s \xrightarrow{o!}^S s'$

and $s \xrightarrow{d}^S s''$ then $d = 0$. Finally, S verifies the *independent progress* condition iff either $(\forall d \geq 0. s \xrightarrow{d}^S)$ or $(\exists d \in \mathbb{R}_{\geq 0}. \exists o! \in \Sigma_o^S . s \xrightarrow{d}^S s'$ and $s' \xrightarrow{o!}^S)$.

TIOTS are syntactically represented by *Timed I/O Automata (TIOA)*. Let X be a finite set of *clocks*. A *clock valuation* over X is a mapping $X \mapsto \mathbb{R}_{\geq 0}$ (thus $\mathbb{R}_{\geq 0}^X$). Given a valuation u and $d \in \mathbb{R}_{\geq 0}$, we write $u + d$ for the valuation in which for each clock $x \in X$ we have $(u+d)(x) = u(x) + d$. For $Y \subseteq X$, we write $u[Y \mapsto 0]$ for a valuation agreeing with u on clocks in $X \setminus Y$, and giving 0 for clocks in Y.

Let $\mathcal{C}(X)$ denote all *clock constraints* φ generated by the grammar $\psi ::= x \prec k \mid x - y \prec k \mid \psi \wedge \psi$, where $k \in \mathbb{Q}$, $x, y \in X$ and $\prec \in \{<, \leq, >, \geq\}$. For $\varphi \in \mathcal{C}(X)$ and $u \in \mathbb{R}_{\geq 0}^X$, we write $u \models \varphi$ if u satisfies φ. Let $[\![\varphi]\!]$ denote the set of valuations $\{u \in \mathbb{R}_{\geq 0}^X \mid u \models \varphi\}$. A subset $Z \subseteq \mathbb{R}_{\geq 0}^X$ is a *zone* if $Z = [\![\varphi]\!]$ for some $\varphi \in \mathcal{C}(X)$.

Definition 2. *A* Timed I/O Automaton *is a tuple* $\mathcal{A} = (Loc, q_0, Clk, E, Act, Inv)$*, where* Loc *is a finite set of* locations*,* $q_0 \in Loc$ *is the* initial location*,* Clk *is a finite set of* clocks*,* $E \subseteq Loc \times Act \times \mathcal{C}(Clk) \times 2^{Clk} \times Loc$ *is a set of* edges*,* $Act = Act_{\rm i} \oplus Act_{\rm o}$ *is a finite set of* actions*, partitioned into* inputs *(*$Act_{\rm i}$*) and* outputs *(*$Act_{\rm o}$*),* $Inv : Loc \mapsto \mathcal{C}(Clk)$ *is a set of location* invariants*. Without loss of generality we assume that invariants of a location are always included in the guards of the edges that are incident with the location. We also assume that guards are satisfiable (for any guard* φ *the set* $[\![\varphi]\!]$ *is non-empty).*

A *universal location*, denoted $l_{\rm u}$, in a TIOA accepts every input and can produce every output at any time. Location $l_{\rm u}$ models an unpredictable behavior of a component.

The semantics of a TIOA $\mathcal{A} = (Loc, q_0, Clk, E, Act, Inv)$ is a TIOTS $[\![\mathcal{A}]\!]_{\rm sem} = (Loc \times \mathbb{R}_{\geq 0}^{Clk}, (q_0, \mathbf{0}), Act, \rightarrow)$, where $\mathbf{0}$ is a constant function mapping all clocks to zero, and $\rightarrow$ is the largest transition relation generated by the following rules:

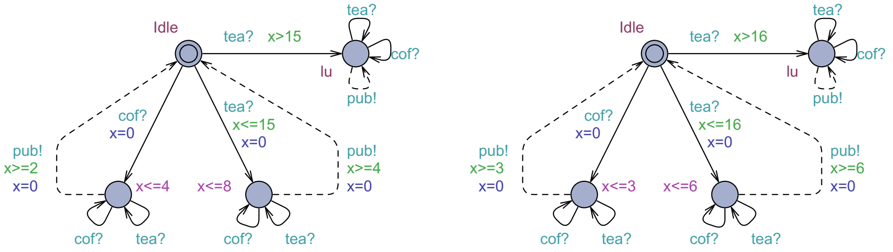
- Each edge $(q, a, \varphi, \lambda, q') \in E$ gives rise to $(q, u) \xrightarrow{a} (q', u')$ for each clock valuation $u \in \mathbb{R}_{\geq 0}^{Clk}$ such that $u \models \varphi$ and $u' = u[\lambda \mapsto 0]$ and $u' \models Inv(q')$.
- Each location $q \in Loc$ with a valuation $u \in \mathbb{R}_{\geq 0}^{Clk}$ gives rise to a transition $(q, u) \xrightarrow{d} (q, u + d)$ for each delay $d \in \mathbb{R}_{\geq 0}$ such that $u + d \models Inv(q)$.

Since TIOTSs are infinite size they cannot be directly manipulated by computations. Usually *symbolic representations*, such as *region graphs* [3] or *zone graphs*, are used as data structures that finitely represent semantics of TIOAs. Let M be the greatest (in absolute value) integer constant that appears in the guards of a TIOA[1]. A *clock region* is an equivalence class of the relation $\sim$ on clock valuations such that $u \sim v$ iff the following conditions hold:

- $\forall x \in Clk$, either $\lfloor u(x) \rfloor = \lfloor v(x) \rfloor$, or $u(x) > M$ and $v(x) > M$,
- $\forall x, y \in Clk$, $\forall c \in [-M, M]$, $u(x) - u(y) \leq c$ iff $v(x) - v(y) \leq c$,
- $\forall x \in Clk$ if $u(x) \leq M$ then $\langle u(x) \rangle = 0$ iff $\langle v(x) \rangle = 0$,

We write $r\nearrow$ for the direct time successor of the region r, if such exists. The *region graph* of a TIOA $\mathcal{A}$ is $\mathcal{G} = (\mathcal{R}_{\mathcal{A}}, \rightarrow^G)$, where $\mathcal{R}_{\mathcal{A}} = \{(q, r) \mid \exists (q, u) \in St^{[\![\mathcal{A}]\!]_{\rm sem}}. u \in r\}$ is the set of regions, and $\rightarrow^G \subseteq \mathcal{R}_{\mathcal{A}} \times (Act \cup \{\tau\}) \times \mathcal{R}_{\mathcal{A}}$, such that $(q, r) \xrightarrow{\tau}^G (q, r\nearrow)$ iff $r\nearrow \models Inv(q)$, and $(q, r) \xrightarrow{a}^G (q', r')$ iff $(q, u) \xrightarrow{a} (q', u')$ for some $u \in r$ and $u' \in r'$.

[1] The region graph of an automaton with rational constants can be built by multiplying all constants of the automaton to work only with integers.



(a) A specification (b) An implementation

Fig. 1. TIOAs of a specification and an implementation for a researcher

Basics of the Timed Specification Theory. In [10], timed specifications and implementations are both represented by TIOAs satisfying additional conditions:

Definition 3. *A* specification $\mathcal{S}$ *is a TIOA whose semantics* $[\![\mathcal{S}]\!]_{\text{sem}}$ *is deterministic and input-enabled. An* implementation $\mathcal{I}$ *is a specification whose semantics* $[\![\mathcal{I}]\!]_{\text{sem}}$ *additionally verifies the output urgency and the independent progress conditions.*

Example 1. Figure 1a presents a specification of a researcher. It accepts either coffee (coff) or tea in order to produce publications (pub). If tea is served after a too long period the researcher falls into an error state represented by a universal state l_{u}. An individual researcher is an implementation of this specification. One example is presented in Figure 1b.

In specification theories, a *refinement* relation plays a central role. It allows to compare specifications, and to relate implementations to specifications. In [10], as well as in [1,2,7], refinement is defined in the style of alternating (timed) simulation:

Definition 4 (Refinement). *An* alternating timed simulation *between TIOTS* $S = (St^S, s_0, \Sigma^S, \rightarrow^S)$ *and* $T = (St^T, t_0, \Sigma^T, \rightarrow^T)$ *is a relation* $R \subseteq St^S \times St^T$ *such that* $(s_0, t_0) \in R$ *and for every* $(s, t) \in R$

- *If* $t \xrightarrow{i?}^T t'$ *for some* $t' \in St^T$*, then* $s \xrightarrow{i?}^S s'$ *and* $(s', t') \in R$ *for some* $s' \in St^S$
- *If* $s \xrightarrow{o!}^S s'$ *for some* $s' \in St^S$*, then* $t \xrightarrow{o!}^T t'$ *and* $(s', t') \in R$ *for some* $t' \in St^T$
- *If* $s \xrightarrow{d}^S s'$ *for* $d \in \mathbb{R}_{\geq 0}$*, then* $t \xrightarrow{d}^T t'$ *and* $(s', t') \in R$ *for some* $t' \in St^T$

We write $S \leq T$ *if there exists an alternating simulation between* S *and* T*. For two TIOAs* $\mathcal{S}$ *and* $\mathcal{T}$*, we say that* $\mathcal{S}$ refines $\mathcal{T}$*, written* $\mathcal{S} \leq \mathcal{T}$*, iff* $[\![\mathcal{S}]\!]_{\text{sem}} \leq [\![\mathcal{T}]\!]_{\text{sem}}$.

Definition 5 (Satisfaction). *An implementation* $\mathcal{I}$ satisfies *a specification* $\mathcal{S}$*, denoted* $\mathcal{I}$ **sat** $\mathcal{S}$*, iff* $[\![\mathcal{I}]\!]_{\text{sem}} \leq [\![\mathcal{S}]\!]_{\text{sem}}$*. We write* $[\![\mathcal{S}]\!]_{\text{mod}}$ *for the set of all implementations of* $\mathcal{S}$*, so* $[\![\mathcal{S}]\!]_{\text{mod}} = \{\mathcal{I} \mid \mathcal{I}$ **sat** $\mathcal{S}$ *and* $\mathcal{I}$ *is an implementation*$\}$.

The reader might find it surprising that in a robust specification theory we refrain from adjusting the refinement to account for imprecision of implementations when comparing specifications. Our basic assumption is that specifications are precise mathematical objects that are not susceptible to imprecision of execution. In contrary, implementations can behave imprecisely when executed, so in Section 3 we will introduce an extension of Def. 5 that takes this into account. It is a fortunate property of Def. 4 that we do not need to modify it in order to reason about robust implementations (Property 3 in Sect. 3).

In [10], we have reduced refinement checking to finding winning strategies in timed games. In the reminder of this section, we recall the definition of such games and show how they can be used to check consistency. Timed games also underly other operations such as conjunction, composition, and quotient [10], which will be illustrated in Sect. 4–5.

Timed Games for Timed I/O Specifications. TIOAs are interpreted as two-player real-time games between the *output player* (the component) and the *input player* (the environment). The input plays with actions in Act_{i} and the output plays with actions in Act_{o}. A strategy for a player is a function that defines her move at a certain time (either delaying or playing a controllable action). A strategy is called *memoryless* if the next move depends solely on the current state. We only consider memoryless strategies, as these suffice for safety games. For simplicity, we only define winning strategies for the output player (i.e. output is the verifier). Definitions for the input player are obtained symmetrically.

Definition 6. *A memoryless* strategy f *for the output player on the TIOA* $\mathcal{A}$ *is a function* $St^{[\![\mathcal{A}]\!]_{\mathrm{sem}}} \mapsto Act_{\mathrm{o}} \cup \{\mathsf{delay}\}$, *such that whenever* $f(s) \in Act_{\mathrm{o}}$ *then* $s \xrightarrow{f(s)} s'$ *for some* s', *and whenever* $f(s) = \mathsf{delay}$ *then* $s \xrightarrow{d} s''$ *for some* $d > 0$ *and state* s''.

The restricted behavior of the TIOA when one player applies a specific strategy is defined as the *outcome* of the strategy.

Definition 7. *Let* $\mathcal{A}$ *be a TIOA,* f *a strategy over* $\mathcal{A}$ *for the output player, and* s *a state of* $[\![\mathcal{A}]\!]_{\mathrm{sem}}$. *The outcome* $\mathsf{Outcome}_{\mathrm{o}}(s, f)$ *of* f *from* s *is the subset of* $\mathsf{Runs}(s, [\![\mathcal{A}]\!]_{\mathrm{sem}})$ *defined inductively by:*

- $s \in \mathsf{Outcome}_{\mathrm{o}}(s, f)$,
- *if* $\rho \in \mathsf{Outcome}_{\mathrm{o}}(s, f)$ *then* $\rho' = \rho \xrightarrow{a} s' \in \mathsf{Outcome}_{\mathrm{o}}(s, f)$ *if* $\rho' \in \mathsf{Runs}(s, [\![\mathcal{A}]\!]_{\mathrm{sem}})$ *and one the following conditions hold:*
  1. $a \in Act_{\mathrm{i}}$,
  2. $a \in Act_{\mathrm{o}}$ *and* $f(\mathsf{last}(\rho)) = a$,
  3. $a \in \mathbb{R}_{\geq 0}$ *and* $\forall d \in [0, a[. \exists s''. \mathsf{last}(\rho) \xrightarrow{d} s''$ *and* $f(s'') = \mathsf{delay}$.
- $\rho \in \mathsf{Outcome}_{\mathrm{o}}(s, f)$ *if* ρ *infinite and all its finite prefixes are in* $\mathsf{Outcome}_{\mathrm{o}}(s, f)$.

A *winning condition* for a player in the TIOA $\mathcal{A}$ is a subset of $\mathsf{Runs}([\![\mathcal{A}]\!]_{\mathrm{sem}})$. In safety games the winning condition is to avoid a set Bad of "bad" states (without lost of generality we assume these "bad" states correspond to a set of entirely "bad" locations). Formally, the winning condition is $WS^{\mathrm{o}}(\mathsf{Bad}) = \{\rho \in \mathsf{Runs}([\![\mathcal{A}]\!]_{\mathrm{sem}}) \mid \mathsf{States}(\rho) \cap \mathsf{Bad} = \emptyset\}$. A strategy f for output is *winning* from state s if $\mathsf{Outcome}_{\mathrm{o}}(s, f) \subseteq WS^{\mathrm{o}}(\mathsf{Bad})$. A state s is winning if there exists a winning strategy from s. The game $(\mathcal{A}, WS^{\mathrm{o}}(\mathsf{Bad}))$ is winning if the initial state is winning. Solving this game is decidable [13,8,10].

Strategies can also be defined symbolically, using the region graph introduced in the previous section. For a region (q, r), if $f(q, r) = \mathsf{delay}$ then $(q, r) \xrightarrow{\tau}^{G} (q, r\nearrow)$, and if $f(q, r) \in Act_{\mathrm{o}}$ then $\exists (q', r'). (q, r) \xrightarrow{a}^{G} (q', r')$. An outcome of f is then a run in the region graph, such that if $\rho \in \mathsf{Outcome}_{o}((q, r), f)$ then $\rho' = \rho \xrightarrow{a} (q', r') \in \mathsf{Outcome}_{o}((q, r), f)$ if $\mathsf{last}(\rho) \xrightarrow{a}^{G} (q', r')$, and either $a \in Act_{\mathrm{i}}$, or $a \in Act_{\mathrm{o}}$ and $f(\mathsf{last}(\rho))$ $= a$, or $a = \tau$ and $f(\mathsf{last}(\rho)) = \mathsf{delay}$.

Maximum Strategies in Timed Games as Operators on Timed Specifications. We sketch how timed games can be used to establish consistency of a timed specifications.

An *immediate error* occurs in a state of a specification if the specification disallows progress of time and output transitions in a given state—such a specification will break if the environment does not send an input. For a specification $\mathcal{S}$ we define the set of immediate error states $\mathsf{err}^{\mathcal{S}} \subseteq \mathit{St}^{[\![\mathcal{S}]\!]_{\mathrm{sem}}}$ as:

$$\mathsf{err}^{\mathcal{S}} = \{ s \mid (\exists d.\, s \not\xrightarrow{d}) \text{ and } \forall d \forall o! \forall s'.\, s \xrightarrow{d} s' \text{ implies } s' \not\xrightarrow{o!} \}$$

It follows that no immediate error states can occur in implementations, since they verify independent progress. In [10] we show that $\mathcal{S}$ is consistent iff there exists a winning strategy for output in the safety game $(\mathcal{S}, \mathit{WS}^{\mathrm{o}}(\mathsf{err}^{\mathcal{S}}))$. Moreover, the maximum consistent part of $\mathcal{S}$ corresponds to the maximum wining strategy for output in this game.

Conjunction of two specifications is found as a maximal strategy for output in a safety game on the product state space to avoid immediate errors. Similarly, optimistic parallel composition of two specifications is computed as a maximum strategy for input in a safety game over the product state space; and a quotient of two specifications is found as a maximum strategy for output in another safety game. Optimistic composition means that two specifications are compatible if there exists at least one environment, in which they can avoid error states. Details can be found in [10].

3 Robust Timed I/O Specifications

We now define a robust extension of our specification theory. An essential requirement for an implementation is to be realizable on a physical hardware, but this requires admitting small imprecisions characteristic for physical components (computer hardware, sensors and actuators). The requirement of realizability has already been linked to the robustness problem in [18] in the context of model checking. In specification theories the small deficiencies of hardware can be reflected in a strengthened satisfaction relation, which introduces small perturbations to the timing of implementation actions, before they are checked against the requirements of a specification—ensuring that the implementation satisfies the specification even if its behavior is perturbed.

We first formalize the concept of perturbation. Let $\varphi \in \mathcal{C}(X)$ be a guard over the set of clocks X. The *enlarged guard* $\lceil \varphi \rceil_\Delta$ is constructed according to the following rules:

- Any term $x_i \prec n_i$ of φ with $\prec \in \{<, \leq\}$ is replaced by $x_i \prec n_i + \Delta$
- Any term $x_i \succ n_i$ of φ with $\succ \in \{>, \geq\}$ is replaced by $x_i \succ n_i - \Delta$

Similarly, the *restricted guard* $\lfloor \varphi \rfloor_\Delta$ is using the two following rules:

- Any term $x_i \prec n_i$ of φ with $\prec \in \{<, \leq\}$ is replaced by $x_i \prec n_i - \Delta$
- Any term $x_i \succ n_i$ of φ with $\succ \in \{>, \geq\}$ is replaced by $x_i \succ n_i + \Delta$.

Notice that for a for a clock valuation u and a guard φ, we have that $u \models \varphi$ implies $u \models \lceil \varphi \rceil_\Delta$, and $u \models \lfloor \varphi \rfloor_\Delta$ implies $u \models \varphi$, and $\lfloor \lceil \varphi \rceil_\Delta \rfloor_\Delta = \lceil \lfloor \varphi \rfloor_\Delta \rceil_\Delta = \varphi$.

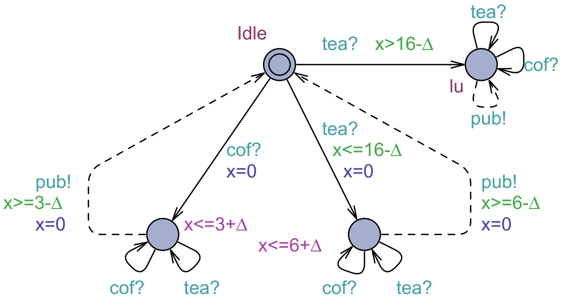


Fig. 2. Δ-perturbation of the researcher implementation

Perturbed Implementation and Robust Timed I/O Specifications. We lift the perturbation to implementation TIOAs. Given a jitter Δ, the perturbation means a Δ-enlargement of invariants and of output edge guards. Guards on the input edges are restricted by Δ:

Definition 8. *For an implementation* $\mathcal{I} = (Loc, q_0, Clk, E, Act, Inv)$ *and* $\Delta \in \mathbb{Q}_{>0}$*, the* Δ-perturbation *of* $\mathcal{I}$ *is the TIOA* $\mathcal{I}_\Delta = (Loc \cup \{l_\mathrm{u}\}, q_0, Clk, E', Act, Inv')$*, such that:*

- *Every edge* $(q, o!, \varphi, \lambda, q') \in E$ *is replaced by* $(q, o!, \lceil\varphi\rceil_\Delta, \lambda, q') \in E'$*,*
- *Every edge* $(q, i?, \varphi, \lambda, q') \in E$ *is replaced by* $(q, i?, \lfloor\varphi\rfloor_\Delta, \lambda, q') \in E'$*,*
- *Every invariant* $Inv(q)$ *is replaced by* $Inv'(q) = \lceil Inv(q)\rceil_\Delta$*,*
- $\forall q \in Loc. \forall i? \in Act_\mathrm{i}.\ (q, i?, \varphi_\mathrm{u}, \emptyset, l_\mathrm{u}) \in E'$ *with* $\varphi_\mathrm{u} = \neg \bigvee_{(q,i?,\varphi,\lambda,q') \in E} \lfloor\varphi\rfloor_\Delta$*.*

$\mathcal{I}_\Delta$ is not necessarily action deterministic, as output guards are enlarged. However it is input-enabled, since by construction (the last case above), any input not accepted after restricting input guards is redirected to the universal location l_u. Also $\mathcal{I}_0$ equals $\mathcal{I}$.

In essence, we weaken the constraints on output edges, and strengthen the constraints on input edges. This is consistent with the game semantics of specifications: perturbation makes the game harder to win for the verifier. Since the gaps created by strengthening input guards are closed by edges to the universal location, the implementation becomes less predictable. If an input arrives close to the deadline, the environment cannot be certain if it will be handled precisely as specified. Enlargement of output guards has a similar effect. The environment of the specification has to be ready that outputs will arrive slightly after the deadlines.

Such considerations are out of place in classical robustness theories for model checking, but are crucial when moving to models, where input and output transitions are distinguished. For example, in [18] the authors propose a robust semantics for timed automata. Their *maximal progress assumption* is equivalent to the output urgency condition of our implementations. However, in [18] both input and output guards are increased, which is suitable for the one-player setting, but incompatible with the contravariant nature of two-player games. Such enlargement would not be monotonic with respect to the alternating refinement (Def. 4), while the perturbation of Def. 8 is monotonic.

We are now ready to define our notion of robust satisfaction:

Definition 9. *An implementation $\mathcal{I}$ robustly satisfies a specification $\mathcal{S}$ given a delay $\Delta \in \mathbb{Q}_{\geq 0}$, denoted $\mathcal{I}$ $\mathbf{sat}_\Delta$ $\mathcal{S}$, iff $\mathcal{I}_\Delta \leq \mathcal{S}$. We write $[\![\mathcal{S}]\!]^{\Delta}_{\text{mod}}$ for the set of all Δ-robust implementations of $\mathcal{S}$, such that $[\![\mathcal{S}]\!]^{\Delta}_{\text{mod}} = \{\mathcal{I} \mid \mathcal{I}\ \mathbf{sat}_\Delta\ \mathcal{S}\ \wedge\ \mathcal{I}$ is an implementation$\}$.*

Property 1. Let $\mathcal{I}$ be an implementation and $0 \leq \Delta_1 < \Delta_2$. Then $\mathcal{I} \leq \mathcal{I}_{\Delta_1} \leq \mathcal{I}_{\Delta_2}$.

In addition, we obtain the following by transitivity of the refinement:

Property 2. Let $\mathcal{S}$ be a specification and $\Delta_1 \leq \Delta_2$, then $[\![\mathcal{S}]\!]^{\Delta_2}_{\text{mod}} \subseteq [\![\mathcal{S}]\!]^{\Delta_1}_{\text{mod}} \subseteq [\![\mathcal{S}]\!]_{\text{mod}}$.

Property 3. Let $\mathcal{S}$ and $\mathcal{T}$ be specifications and $0 \leq \Delta$, then $\mathcal{S} \leq \mathcal{T} \implies [\![\mathcal{S}]\!]^{\Delta}_{\text{mod}} \subseteq [\![\mathcal{T}]\!]^{\Delta}_{\text{mod}}$.

We now turn to the problem of deciding whether a specification is robustly consistent:

Definition 10. *Let $\mathcal{S}$ be a specification and $\Delta \in \mathbb{Q}_{>0}$, then $\mathcal{S}$ is Δ-*robust consistent *if there exists an implementation $\mathcal{I}$ such that $\mathcal{I}$ $\mathbf{sat}_\Delta$ $\mathcal{S}$.*

Like in the non-robust case, deciding consistency and performing operations on specifications are reducible to solving games. But now, we will need to make the games aware of the robustness conditions. In the rest of this section, we propose a definition for such games. Then, in Sections 4 and 5, we show how they can be used to perform classical operations on specifications.

Example 2. Figure 2 presents the Δ-perturbation of the researcher implementation presented in Fig. 1b. One can check that this implementation robustly satisfies the specification of Fig. 1a for any $\Delta \in]0, 1]$.

Robust Timed Games for Timed I/O Specifications. We first define robust strategies that guarantee winning even if subject to bounded timing perturbations. We then propose a technique for finding such strategies. We start with the construction of *syntactic outcome* that represents game outcomes as TIOAs. We rely on the region graph construction in the definition of syntactic outcome, but any stable partitioning of the state-space could serve this purpose, and would be more efficient in practice.

Definition 11. *Let $\mathcal{A} = (Loc, q_0, Clk, E, Act, Inv)$ be a TIOA and f a strategy over $\mathcal{A}$ for output. The TIOA $\mathcal{A}_f = (\mathcal{R}_\mathcal{A}, (q_0, r_0), Clk \cup \{z\}, \widehat{E}, Act \cup \{\tau\}, \widehat{Inv})$ is built by decorating the region graph $\mathcal{G} = (\mathcal{R}_\mathcal{A}, \rightarrow^G)$ of $\mathcal{A}$, using the original clocks of $\mathcal{A}$ and an additional clock z to impose output urgency. For each region (q, r), the incident edges and the invariant are defined as follows:*

- *$\widehat{Inv}(q, r) = Inv(q) \wedge (r \vee r\nearrow)$;*
- *If $(q, r) \xrightarrow{\tau}^G (q, r\nearrow)$ then $((q, r), \tau, r\nearrow, \{z\}, (q, r\nearrow)) \in \widehat{E}$;*
- *For each edge $(q, i?, \varphi, \lambda, q') \in E$, if $(q, r) \xrightarrow{i?}^G (q', r')$ then $((q, r), i?, \varphi, \lambda \cup \{z\}, (q', r')) \in \widehat{E}$;*
- *If $f(q, r) = o!$, then $\widehat{Inv}(q, r) = r \wedge \{z = 0\}$, and for each edge $(q, o!, \varphi, \lambda, q') \in E$, if $(q, r) \xrightarrow{o!}^G (q', r')$, then $((q, r), o!, \varphi, \lambda \cup \{z\}, (q', r')) \in \widehat{E}$.*

This construction captures the semantic outcome of the game in the following sense:

Proposition 1. *Let $(\mathcal{A}, W)$ be a timed safety game and f be a strategy for output. Then a run ρ in the region graph $\mathcal{G}$ of $\mathcal{A}$ is in* $\mathsf{Outcome}_\mathrm{o}((q,r), f)$ *iff ρ is an untimed run of $\mathcal{A}_f$ starting from region (q,r).*

In a robust timed game we seek strategies that remain winning after perturbation by a delay Δ. The perturbation is defined on the syntactic outcome of the strategy, by enlarging the guards for the actions of the verifier. We write $\lceil \mathcal{A} \rceil^\mathrm{o}_\Delta$ (resp. $\lceil \mathcal{A} \rceil^\mathrm{i}_\Delta$) for the TIOA where the guards of the output (resp. input) player and the invariants have been enlarged by Δ.

Definition 12. *For a timed game $(\mathcal{A}, W)$, a strategy f for output is Δ-*robust winning *if it is winning when the moves of output are perturbed, i.e.* $\mathsf{Runs}([\![\lceil \mathcal{A}_f \rceil^\mathrm{o}_\Delta]\!]_\mathrm{sem}) \subseteq W$.[2]

As proposed in [9], robust timed games for a bounded delay can be reduced to classical timed games by a syntactic transformation of the game automaton. Here, we propose a modified version of the construction that respects the duality between inputs and outputs:

Definition 13. *Let $\mathcal{A} = (Loc, q_0, Clk, E, Act, Inv)$ be TIOA and a $\Delta \in \mathbb{Q}_{>0}$, the* robust game automaton $\mathcal{A}^\Delta_\mathrm{rob} = (\widetilde{Loc}, q_0, Clk \cup \{y\}, \widetilde{E}, \widetilde{Act}, \widetilde{Inv})$ *uses an additional clock y and is constructed according to the following rules:*

- *$Loc \subseteq \widetilde{Loc}$, and for each location $q \in Loc$ and each edge $e = (q, o!, \varphi, \lambda, q') \in E$, two locations q_e^α and q_e^β are added in $\widetilde{Loc}$. The invariant of q is unchanged; the invariants of q_e^α and q_e^β are $y \leq \Delta$.*
- *For each action $o! \in Act_\mathrm{o}$, an additional action $o?$ is added in $\widetilde{Act}_\mathrm{i}$.*
- *Each edge $e' = (q, i?, \varphi, \lambda, q') \in E$ gives rise to the following edges in $\widetilde{E}$: $(q, i?, \varphi, \lambda \cup \{y\}, q')$, $(q_e^\alpha, i?, \varphi, \lambda \cup \{y\}, q')$ and $(q_e^\beta, i?, \varphi, \lambda \cup \{y\}, q')$.*
- *Each edge $e = (q, o!, \varphi, \lambda, q') \in E$ gives rise to the following edges in $\widetilde{E}$: $(q, o!, \varphi\rfloor_{2\Delta}, \{y\}, q_e^\alpha)$, $(q_e^\alpha, o!, \{y = \Delta\}, \{y\}, q_e^\beta)$, $(q_e^\alpha, o?, \{y \leq \Delta\}, \lambda \cup \{y\}, q')$ and $(q_e^\beta, o?, \{y \leq \Delta\}, \lambda \cup \{y\}, q')$, where $\varphi\rfloor_{2\Delta}$ is constructed from φ by restricting upper bound constraints by 2Δ.*

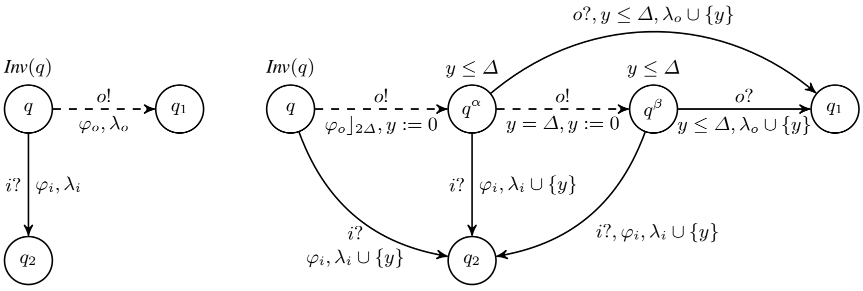
The construction is demonstrated in Fig 3. The output player can propose a move in the time interval $\lfloor \varphi \rfloor_\Delta$ (which is done in two steps: first playing $o!$ in the time interval $\varphi\rfloor_{2\Delta}$ and then a second firing after Δ time units), but the input player can perturb this move by choosing a smaller or greater delay to perform the action.

The construction shall serve as a tool for deciding robust consistency, synthesizing a robust implementation, and other operations of the specification theory with robustness.

Theorem 1. *For a timed safety game $(\mathcal{A}, W)$, if f is a winning strategy for output in the robust game $(\mathcal{A}^\Delta_\mathrm{rob}, W)$, then the following strategy f' is a Δ-robust winning strategy for output in the game $(\mathcal{A}, W)$: $\forall (q,r) \in \mathcal{R}_\mathcal{A}$, $f'(q,r) = o!$ if $\exists e. f(q_e^\alpha, \widetilde{r}) = o!$, where $\widetilde{r}$ is a region of $\mathcal{R}_{\mathcal{A}^\Delta_\mathrm{rob}}$, and r its projection on $\mathcal{R}_\mathcal{A}$*[3]*, otherwise $f'(q,r) =$* delay.

[2] Technically, we assume that runs in $\mathsf{Runs}([\![\lceil \mathcal{A}_f \rceil^\mathrm{o}_\Delta]\!]_\mathrm{sem})$ abstract τ transitions

[3] To this end the region graph of $\mathcal{A}$ must be computed with Δ as the lowest constant.


$Inv(q)$
q
$o!$
φ_o, λ_o
q_1
$i?$
φ_i, λ_i
q_2
$Inv(q)$
q
$o!$
$\varphi_o\rfloor_{2\Delta}, y := 0$
$y \leq \Delta$
q^α
$o!$
$y = \Delta, y := 0$
$y \leq \Delta$
q^β
$o?$
$y \leq \Delta, \lambda_o \cup \{y\}$
$o?, y \leq \Delta, \lambda_o \cup \{y\}$
q_1
$i?$
$\varphi_i, \lambda_i \cup \{y\}$
$i?$
$\varphi_i, \lambda_i \cup \{y\}$
$i?, \varphi_i, \lambda_i \cup \{y\}$
q_2

(a) TIOA $\mathcal{A}$

(b) Robust game automaton $\mathcal{A}^{\Delta}_{\text{rob}}$

Fig. 3. Construction of the robust game automation $\mathcal{A}^{\Delta}_{\text{rob}}$ from an original automaton $\mathcal{A}$

Our method and our notion of robust strategy guarantees non-zenoness for the synthesized strategies. Indeed by allowing the opponent to perturb the verifier by Δ or $-\Delta$ we impose that the verifier only performs non-null delay actions. Later, in the context of timed specifications, this ensures realizability of implementations. Also non-Zeno environements are used as witnesses of compatibility in optimistic composition, which could have happened in [10], which ignored Zeno-problems altogether.

4 Robust Consistency and Conjunction

We now provide a method to decide the Δ-robust consistency of a specification and synthesize robust implementations by solving a robust timed safety game, in which the output player must avoid a set of immediate error states. From there, the computation of a robust strategy gives a method to synthesize a robust implementation of the specification.

Intuitively a specification is Δ-robust with respect to input $i?$, if between enabling of any two $i?$ edges at least 2Δ time passes, during which the reaction to $i?$ is unspecified. So, if the two transitions triggers Δ-too-late and Δ-too-early (respectively), there is no risk that the reaction is resolved non-deterministically in the specification.

In our input-enabled setup, lack of reaction is modeled using transitions to the universal (unpredictable) state. Formally, we say that Δ-robust specifications should admit Δ-*latency* of inputs. A state (q, u) verifies the Δ-latency condition for inputs, iff for each edge $e = (q, i?, \varphi, c, q')$, where $q' \neq l_{\text{u}}$ and e is enabled in (q, u) we have:

$$\begin{aligned}&\forall d \in [0, 2\Delta].\, \forall e' = (q, i?, \varphi, c, q'').\\ &\qquad \text{if } e' \neq e \text{ and } (q, u) \xrightarrow{d} (q, u + d) \text{ and } e' \text{ is enabled in } (q, u + d) \text{ then } q'' = l_{\text{u}}\end{aligned}$$

For a specification $\mathcal{S}$, the safety objective for the robust consistency game is to avoid the set of states of error states $\mathsf{err}^{\mathcal{S}}_{\Delta}$ such that $(q, u) \in \mathsf{err}^{\mathcal{S}}_{\Delta}$ iff (q, u) violates independent progress or Δ-latency for inputs, so:

- Violates independent progress: $(\exists d \in \mathbb{R}_{\geq 0}.(q, u) \overset{d}{\not\to})$ and $(\forall d \forall o!.(q, u) \xrightarrow{d} (q, u + d) \Rightarrow (q, u + d) \overset{o!}{\not\to})$,

- Violates Δ-latency of inputs: $\exists e = (q, i?, \varphi, c, q')$, $q' \neq l_\mathrm{u}$, enabled in (q, u), such that $\exists d \in [0, 2\Delta].(q, u) \xrightarrow{d} (q, u+d)$ and $\exists e' = (q, i?, \varphi, c, q'')$ enabled in $(q, u+d)$, with $e' \neq e$ and $q'' \neq l_\mathrm{u}$.

Observe that $\mathsf{err}^\mathcal{S} \subseteq \mathsf{err}^\mathcal{S}_\Delta$, because the error condition with robustness is weaker than in the classical case (cf. page 135).

The robust safety game $(\mathcal{S}, \mathit{WS}^\mathrm{o}(\mathsf{err}^\mathcal{S}_\Delta))$ can be solved with the method presented in the previous section, and a winning strategy f for the game can be synthesized. Let $\mathcal{S}_f$ be the syntactic outcome of f in $\mathcal{S}$. We build from a robust implementation $\mathcal{I}_f$ by applying the following transformation to $\mathcal{S}_f$:

- When we apply a Δ-perturbation on $\mathcal{I}_f$, a state $((q, r), u)$ can be reached even if $u \notin r \vee r\nearrow$. However due to the region partitioning, the inputs available in $\mathcal{S}_f$ might not be firable from $r \vee r\nearrow$. Then, in order to check the robust satisfaction relation between $(\mathcal{I}_f)_\Delta$ and $\mathcal{S}$, we add additional input edges to $\mathcal{I}_f$: for each location (q, r) in $\mathcal{S}_f$, for each edge $e = ((q, r), i?, \varphi, \lambda \cup \{z\}, (q^*, r^*))$ (with $q^* \neq l_\mathrm{u}$), and for each location (q', r') linked to (q, r) by a sequence of τ transitions, we add an edge $e' = ((q', r'), i?, \varphi, \lambda \cup \{z\}, (q^*, r^*))$.
- To support restriction of input guards in $(\mathcal{I}_f)_\Delta$, we replaced in $\mathcal{I}_f$ all guards φ of edges $e = ((q, r), i?, \varphi, \lambda \cup \{z\}, (q', r'))$ with $q' \neq l_\mathrm{u}$ by their elargement $\lceil \varphi \rceil_\Delta$. Guards on edges to the l_u location are adjusted in order to maintain action determinism and input-enableness.

Note that this construction adds many input edges to the implementation, out of which many are never enabled. This simplifies the construction and the proof of correctness. In practice, to efficiently synthesize implementations, coarser abstractions like zones should be used, that do not include τ transitions and thus avoid multiplying input edges.

Theorem 2. *For a specification $\mathcal{S}$ and a robust winning strategy f in the Δ-robust consistency game, we have that $\mathcal{I}_f$ is a Δ-robust implementation of $\mathcal{S}$, i.e. $\mathcal{I}_f$ $\mathbf{sat}_\Delta$ $\mathcal{S}$.*

Conjunction. A conjunction of two specifications captures the intersection of their implementation sets. The following conjunction operator has been proposed in [10]:

Definition 14. *Let $\mathcal{S} = (\mathit{Loc}^S, q_0^S, \mathit{Clk}^S, E^S, \mathit{Act}, \mathit{Inv}^S)$ and $\mathcal{T} = (\mathit{Loc}^T, q_0^T, \mathit{Clk}^T, E^T, \mathit{Act}, \mathit{Inv}^T)$ be specifications that share the same alphabet of actions Act. We define their conjunction, denoted $\mathcal{S} \wedge \mathcal{T}$, as the TIOA $(\mathit{Loc}, q_0, \mathit{Clk}, E, \mathit{Act}, \mathit{Inv})$ where $\mathit{Loc} = \mathit{Loc}^S \times \mathit{Loc}^T$, $q_0 = (q_0^S, q_0^T)$, $\mathit{Clk} = \mathit{Clk}^S \uplus \mathit{Clk}^T$, $\mathit{Inv}((q_s, q_t)) = \mathit{Inv}(q_s) \wedge \mathit{Inv}(q_t)$, and the set of edges is defined by the following rule: if $(q_s, a, \varphi_s, c_s, q_s') \in E^S$ and $(q_t, a, \varphi_t, c_t, q_t') \in E^T$ then $((q_s, q_t), a, \varphi_s \wedge \varphi_t, c_s \cup c_t, (q_s', q_t')) \in E$.*

It turns out that this operator is robust, in the sense of precisely characterizing also the intersection of the sets of *robust* implementations. So not only conjunction is the greatest lower bound with respect to implementation semantics, but also with respect to the robust implementation semantics. More precisely:

Theorem 3. *For specifications $\mathcal{S}$, $\mathcal{T}$ and $\Delta \in \mathbb{Q}_{>0}$: $[\![\mathcal{S} \wedge \mathcal{T}]\!]^\Delta_\mathrm{mod} = [\![\mathcal{S}]\!]^\Delta_\mathrm{mod} \cap [\![\mathcal{T}]\!]^\Delta_\mathrm{mod}$*

The theorem is a direct extension for robust implemenations of Theorem 6 in [10]. We remark that due to the monotonicity of the refinement (Property 1), we can use two different delays Δ_1 and Δ_2, such that $[\![\mathcal{S}]\!]^{\Delta_1}_{\mathrm{mod}} \cap [\![\mathcal{T}]\!]^{\Delta_2}_{\mathrm{mod}} \supseteq [\![\mathcal{S} \wedge \mathcal{T}]\!]^{\max(\Delta_1,\Delta_2)}_{\mathrm{mod}}$. So requirements with different precision can be conjoined, by considering the smaller jitter.

Robustness of the operator in Def. 14 is very fortunate. Thanks to this large parts of implementation of theory of [10] can be reused. We have experimented on small examples asking simple robust consistency questions by applying constructions manually and using non-robust version of ECDAR [11], obtaining promising results.

5 Robust Compatibility, Composition and Quotient

We lift the composition and quotient operators [10] to the robust setting. Composition is used to build systems from smaller units, while quotient is used to synthesize specifications of missing components in a larger design, for example for controller synthesis.

Parallel Composition. Two specifications $\mathcal{S}, \mathcal{T}$ can be composed only iff $Act_{\mathrm{o}}^{\mathcal{S}} \cap Act_{\mathrm{o}}^{\mathcal{T}} = \emptyset$. Parallel composition is obtained by a product, where the inputs of one specification synchronize with the outputs of the other:

Definition 15 (Parallel Composition). *Let $\mathcal{S} = (Loc^S, q_0^S, Clk^S, E^S, Act^S, Inv^S)$ and $\mathcal{T} = (Loc^T, q_0^T, Clk^T, E^T, Act^T, Inv^T)$ be two composable specifications. We define their parallel product, denoted $\mathcal{S} \parallel \mathcal{T}$, as the TIOA $(Loc, q_0, Clk, E, Act, Inv)$ where $Loc = Loc^S \times Loc^T$, $q_0 = (q_0^S, q_0^T)$, $Clk = Clk^S \uplus Clk^T$, $Inv(q_s, q_t) = Inv(q_s) \wedge Inv(q_t)$, and the set of edges is defined by the three following rules:*

- *if $(q_s, a, \varphi_s, c_s, q_s') \in E^S$ with $a \in Act^S \backslash Act^T$ then each $q_t \in Loc^T$ gives rise to an edge $((q_s, q_t), a, \varphi_s, c_s, (q_s', q_t)) \in E$;*
- *if $(q_t, a, \varphi_t, c_t, q_t') \in E^T$ with $a \in Act^T \backslash Act^S$ then each $q_s \in Loc^S$ gives rise to an edge $((q_s, q_t), a, \varphi_t, c_t, (q_s, q_t')) \in E$;*
- *if $(q_s, a, \varphi_s, c_s, q_s') \in E^S$ and $(q_t, a, \varphi_t, c_t, q_t') \in E^T$ with $a \in Act^S \cap Act^T$ then this gives rise to an edge $((q_s, q_t), a, \varphi_s \wedge \varphi_t, c_s \cup c_t, (q_s'', q_t')) \in E$.*

Robustness distributes over parallel composition in the following fashion:

Lemma 1. *For any implementations $\mathcal{I}$, $\mathcal{J}$ and a delay $\Delta \in \mathbb{Q}_{>0}$: $(\mathcal{I} \parallel \mathcal{J})_\Delta \leq \mathcal{I}_\Delta \parallel \mathcal{J}_\Delta$*

We model incompatibility by introducing a predicate describing undesirable states denoted by the set und. For example, a communication failure in the input-enabled setting can be modeled, by redirecting an input edge to an undesirable location. In general any reachability objective, for example given by a temporal logics property, can serve as the set of undesirable behaviors und. It is important that such behaviors are avoided during the composition. For doing so, we propose to follow the optimistic approach to composition introduced in [1] that is *two specifications can be composed if there exists at least one environment in which they can work together*. In the robustness setting we consider imprecise environments by applying a Δ-perturbation to their outputs. Then, in what follows, we say that a specification is *Δ-robust useful* if there exists an imprecise environment $\mathcal{E}$ that avoids the undesirable states, whatever the specification does.

Definition 16. *A specification $\mathcal{S}$ is* Δ-robust useful *if there exists an environment $\mathcal{E}$ such that no undesirable states are reached in $[\![\lceil\mathcal{E}\rceil^{o}_{\Delta} \parallel \mathcal{S}]\!]_{\text{sem}}$.*

To check robust usefulness we solve the robust game $(\mathcal{S}, \mathit{WS}^{\text{i}}(\mathsf{und}))$, and determine if the input player has a robust strategy f that avoids the undesirable states. Let $\mathcal{S}_f$ be the syntactic outcome of f in $\mathcal{S}$. We build from $\mathcal{S}_f$ a robust environment $\mathcal{E}_f$ by permuting the input and output players, such that each input in $\mathcal{S}_f$ becomes an output, and conversely.

Theorem 4. *If there exists a robust winning strategy f in the Δ-robust usefulness game for a specification $\mathcal{S}$, then $\mathcal{S}$ is Δ-robust useful in the environment $\mathcal{E}_f$.*

Finally, two specifications are compatible if their composition is useful.

Definition 17. *Two composable specifications $\mathcal{S}$ and $\mathcal{T}$ are* Δ-robust compatible *iff $\mathcal{S} \parallel \mathcal{T}$ is Δ-robust useful.*

It is important that the robust theory does not modify the definition of the operations themself. This means that all the important properties of composition introduced in [10] remain valid. This is illustrated with the independent implementability property in Theorem 5, which follows from Lemma 1 and Thm. 10 in [10].

Theorem 5. *Let $\mathcal{S}$ and $\mathcal{T}$ be composable specifications and let $\mathcal{I}$ and $\mathcal{J}$ be Δ-robust implementations of $\mathcal{S}$ and $\mathcal{T}$ (resp.), i.e. $\mathcal{I}$ $\mathbf{sat}_{\Delta}$ $\mathcal{S}$ and $\mathcal{J}$ $\mathbf{sat}_{\Delta}$ $\mathcal{S}$, then $\mathcal{I} \parallel \mathcal{J}$ $\mathbf{sat}_{\Delta}$ $\mathcal{S} \parallel \mathcal{T}$. Moreover if $\mathcal{S}$ and $\mathcal{T}$ are Δ-compatible then $\mathcal{I}$ and $\mathcal{J}$ are also Δ-compatible.*

Due to the monotonicity of perturbations with respect to the refinement, two different delays can be used to implement specifications $\mathcal{S}$ and $\mathcal{T}$. For two implementations $\mathcal{I}$ $\mathbf{sat}_{\Delta_1}$ $\mathcal{S}$ and $\mathcal{J}$ $\mathbf{sat}_{\Delta_2}$ $\mathcal{T}$ of the parallel components, their composition satisfies the composition of specifications with the smaller of the two precisions: $\mathcal{I} \parallel \mathcal{J}$ $\mathbf{sat}_{\min(\Delta_1,\Delta_2)}$ $\mathcal{S} \parallel \mathcal{T}$.

Quotient. Quotient is a dual operator to composition, such that for a large specification $\mathcal{T}$ and a small one $\mathcal{S}$, $\mathcal{T} \setminus\!\!\setminus \mathcal{S}$ is the specification of the components that composed with $\mathcal{S}$ will refine $\mathcal{T}$. In other words, $\mathcal{T} \setminus\!\!\setminus \mathcal{S}$ specifies the component that still needs to be implemented after having an implementation of $\mathcal{S}$, in order to build an implementation of $\mathcal{T}$. One possible application is when $\mathcal{T}$ is a system specification, and $\mathcal{S}$ is the plant, then a robust controller for a safety objective can be achieved by finding a Δ-consistent implementation of the quotient $\mathcal{T} \setminus\!\!\setminus \mathcal{S}$.

To apply quotienting, we require that $\mathit{Act}^S \subseteq \mathit{Act}^T$ and $\mathit{Act}^S_{\text{o}} \subseteq \mathit{Act}^T_{\text{o}}$. The construction of a quotient requires the use of a universal location l_{u}, as well as an inconsistent location $l_\emptyset$ that forbids any outputs and forbids elapsing of time.

Definition 18 (Quotient). *Let $\mathcal{T} = (\mathit{Loc}^T, q^T_0, \mathit{Clk}^T, E^T, \mathit{Act}^T, \mathit{Inv}^T)$ and $\mathcal{S} = (\mathit{Loc}^S, q^S_0, \mathit{Clk}^S, E^S, \mathit{Act}^S, \mathit{Inv}^S)$ with $\mathit{Act}^S \subseteq \mathit{Act}^T$ and $\mathit{Act}^S_{\text{o}} \subseteq \mathit{Act}^T_{\text{o}}$. Their* quotient*, denoted $\mathcal{T} \setminus\!\!\setminus \mathcal{S}$, is the TIOA $(\mathit{Loc}, q_0, \mathit{Clk}, E, \mathit{Act}, \mathit{Inv})$ where $\mathit{Loc} = \mathit{Loc}^T \times \mathit{Loc}^S \cup \{l_{\text{u}}, l_\emptyset\}$, $q_0 = (q^T_0, q^S_0)$, $\mathit{Clk} = \mathit{Clk}^T \uplus \mathit{Clk}^S \uplus \{x_{new}\}$, $\mathit{Act} = \mathit{Act}_{\text{i}} \uplus \mathit{Act}_{\text{o}}$ with $\mathit{Act}_{\text{i}} = \mathit{Act}^T_{\text{i}} \cup \mathit{Act}^S_{\text{o}} \cup \{i_{new}\}$ and $\mathit{Act}^T_{\text{o}} \setminus \mathit{Act}^S_{\text{o}}$, $\mathit{Inv}(q_t, q_s) = \mathit{Inv}(l_{\text{u}}) = \mathit{true}$ and $\mathit{Inv}(l_\emptyset) = \{x_{new} \leq 0\}$, and the set E of edges is defined by the following rules:*

- $\forall q_t \in \mathit{Loc}^T.\forall q_s \in \mathit{Loc}^S.\forall a \in \mathit{Act}.\exists((q_t, q_s), a, \neg \mathit{Inv}^S(q_s), \{x_{new}\}, l_{\mathrm{u}}) \in E$,
- $\forall q_t \in \mathit{Loc}^T.\forall q_s \in \mathit{Loc}^S.\exists((q_t, q_s), i_{new}, \neg \mathit{Inv}(q_t) \wedge \mathit{Inv}(q_s), \{x_{new}\}, l_\emptyset) \in E$,
- *if* $(q_t, a, \varphi_t, c_t, q_t') \in E^T$ *and* $(q_s, a, \varphi_s, c_s, q_s') \in E^S$, *then* $\exists((q_t, q_s), a, \varphi^T \wedge \varphi^S, c_t \cup c_s, (q_t', q_s')) \in E$,
- $\forall(q_s, a, \varphi_s, c_s, q_s') \in E^S$ *with* $a \in \mathit{Act}_{\mathrm{o}}^S$, $\exists((q_t, q_s), a, \varphi^S \wedge \neg G^T, \{x_{new}\}, l_\emptyset) \in E$, *where* $G^T = \bigvee\{\varphi_t \mid (q_t, a, \varphi_t, c_t, q_t')\}$,
- $\forall(q_t, a, \varphi_t, c_t, q_t') \in E^T$ *with* $a \notin \mathit{Act}^S$, $\exists((q_t, q_s), a, \varphi^T, c_t, (q_t', q_s')) \in E$,
- $\forall(q_t, a, \varphi_t, c_t, q_t') \in E^T$ *with* $a \in \mathit{Act}_{\mathrm{o}}^S$, $\exists((q_t, q_s), a, \neg G^T, \{\}, l_{\mathrm{u}}) \in E$, *where* $G^T = \bigvee\{\varphi_s \mid (q_s, a, \varphi_s, c_s, q_s')\}$,
- $\forall a \in \mathit{Act}_{\mathrm{i}}.\exists(l_\emptyset, a, x_{new} = 0, \emptyset, l_\emptyset) \in E$,
- $\forall a \in \mathit{Act}.\exists(l_{\mathrm{u}}, a, true, \emptyset, l_{\mathrm{u}}) \in E$.

As stated in Thm. 12 of [10], the quotient gives a maximal (the weakest) specification for a missing component. This theorem can be generalized to specifications that are locally consistent (see [10]), and used to argue for completeness of the quotient construction in the robust case. It turns out that this very operator is also maximal for the specification of a robust missing component, in the following sense:

Theorem 6. *Let $\mathcal{S}$ and $\mathcal{T}$ be two specifications such that the quotient $\mathcal{T} \mathbin{\backslash\!\backslash} \mathcal{S}$ is defined and let $\mathcal{J}$ be an implementation, then*

$$\mathcal{S} \parallel \mathcal{J}_\Delta \leq \mathcal{T} \quad \textit{iff} \quad \mathcal{J}\, \mathbf{sat}_\Delta\, \mathcal{T} \mathbin{\backslash\!\backslash} \mathcal{S}$$

6 Concluding Remarks

We have presented a compositional framework for reasoning about robustness of timed I/O specifications. Our theory builds on the results presented in [10] combined together with a new robust timed game for robust specification theories. We extend the construction of [9] to the setting of specification theories, to solve robust games by reducing them to problems on classical timed games. This construction can easily be implemented in tools such as ECDAR. Our approach can be used to synthesize an implementation that is robust with respect to a given specification, and to combine or compare specifications in a robust manner. Our approach can potentially be applied to lift any game-based timed specification theory to a robust setting.

In the future, we will consider the parametric extension of our theory, that is to synthesize the value of the perturbation for which consistency holds. The emptiness problem that decides if there exists a robust delay has already been studied in the case of 1-player games in different works [14,17,5,6]. A quantitative analysis of this problem has only been studied in [12].

References

1. de Alfaro, L., Henzinger, T.A.: Interface automata. In: ESEC / SIGSOFT FSE, pp. 109–120 (2001)
2. de Alfaro, L., Henzinger, T.A.: Interface-based design. In: Engineering Theories of Software Intensive Systems, Marktoberdorf Summer School (2004)

3. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
4. Badouel, E., Benveniste, A., Caillaud, B., Henzinger, T., Legay, A., Passerone, R.: Contract theories for embedded systems: A white paper. Research report, IRISA/INRIA Rennes (2009)
5. Bouyer, P., Markey, N., Reynier, P.A.: Robust model-checking of linear-time properties in timed automata. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 238–249. Springer, Heidelberg (2006)
6. Bouyer, P., Markey, N., Reynier, P.A.: Robust analysis of timed automata via channel machines. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 157–171. Springer, Heidelberg (2008)
7. Bulychev, P., Chatain, T., David, A., Larsen, K.G.: Efficient on-the-fly algorithm for checking alternating timed simulation. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 73–87. Springer, Heidelberg (2009)
8. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005)
9. Chatterjee, K., Henzinger, T.A., Prabhu, V.S.: Timed parity games: Complexity and robustness. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 124–140. Springer, Heidelberg (2008)
10. David, A., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: Timed I/O automata: a complete specification theory for real-time systems. In: HSCC, pp. 91–100. ACM, New York (2010)
11. David, A., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: ECDAR: An environment for compositional design and analysis of real time systems. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 365–370. Springer, Heidelberg (2010)
12. Jaubert, R., Reynier, P.A.: Quantitative robustness analysis of flat timed automata. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 229–244. Springer, Heidelberg (2011)
13. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: Finkel, A., Jantzen, M. (eds.) STACS 1992. LNCS, vol. 577, pp. 229–242. Springer, Heidelberg (1992)
14. Puri, A.: Dynamical properties of timed automata. In: Ravn, A.P., Rischel, H. (eds.) FTRTFT 1998. LNCS, vol. 1486, pp. 210–227. Springer, Heidelberg (1998)
15. The COMBEST Consortium: Combest, http://www.combest.eu.com
16. The SPEEDS Consortium: Speeds, http://www.speeds.eu.com
17. Wulf, M., Doyen, L., Markey, N., Raskin, J.F.: Robust safety of timed automata. Formal Methods in System Design 33, 45–84 (2008)
18. Wulf, M.D., Doyen, L., Raskin, J.F.: Almost ASAP semantics: from timed models to timed implementations. Formal Aspects of Computing 17(3), 319–341 (2005)

Minimum Attention Controller Synthesis for Omega-Regular Objectives*

Krishnendu Chatterjee[1] and Rupak Majumdar[2]

[1] IST Austria
[2] MPI-SWS

Abstract. A controller for a discrete game with ω-regular objectives requires *attention* if, intuitively, it requires measuring the state and switching from the current control action. Minimum attention controllers are preferable in modern shared implementations of cyber-physical systems because they produce the least burden on system resources such as processor time or communication bandwidth. We give algorithms to compute minimum attention controllers for ω-regular objectives in imperfect information discrete two-player games. We show a polynomial-time reduction from minimum attention controller synthesis to synthesis of controllers for mean-payoff parity objectives in games of incomplete information. This gives an optimal EXPTIME-complete synthesis algorithm. We show that the minimum attention controller problem is decidable for infinite state systems with finite bisimulation quotients. In particular, the problem is decidable for timed and rectangular automata.

1 Introduction

Automata-theoretic reactive synthesis techniques [10,5,31,15,32,29,37,21,23] hold the promise to correct-by-construction design of complex reactive systems, and over the years, have seen impressive technical advances that have brought them within striking distance of practice in the design of cyber-physical systems [17,19,20,26]. Despite the many advances, there is still a gap between the *abstract* two-person game models considered by the theory of synthesis and *implementation issues* associated with controller implementations. Classically, automata-theoretic synthesis considers the size of the memory as the notion of optimality; and any memoryless strategy is considered optimal (a strategy is *memoryless* if it depends only on the current state and not on the history of the play). While roughly adequate for applications of synthesis in hardware circuit design, for more recent applications of synthesis techniques in cyber-physical systems, there are additional implementation costs whose effects can significantly influence the practical applicability of synthesized controllers.

Classical controller synthesis assumes that measurement of the current state and computation of the control action is computed instantaneously. In practice, state measurement and control computations take time and consume other system resources such

* This research was funded in part by the US National Science Foundation grants CCF-0546170, CNS-0702881, DARPA grant HR0011-09-1-0037, Austrian Science Fund (FWF) NFN Grant S11407-N23 (RiSE), and a Microsoft faculty fellowship.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 145–159, 2011.

as network bandwidth. In a modern control system application, where a controller shares the platform with other tasks, the controller task must compete for resources with other tasks. In this context, it is important to find a controller that can be implemented without diverting attention from other, possibly more pressing, system tasks, or to enable other tasks which might not be schedulable if the controller hogs resources.

Consider, for example, the simple game in Figure 1 where the objective is to visit the state s. For states in $\{1,\ldots,n\}$, any function $\xi : \{1,\ldots,n\} \to \{0,1\}$ is a memoryless winning strategy ensuring a visit to s, and equally good in the view of classical synthesis. However, consider a real-time implementation of the controller where the control task must be scheduled to compute the next control action. The strategy which always plays 0 (or 1) has an advantage over the strategy playing the action $i \mod 2$. For the former strategy, the controller task is scheduled once to set the control action, and never again. For the latter, the control task runs every cycle, looking at the state and changing the control action accordingly, using up communication resources to measure state and processor resources to compute the new control, which could be used for other tasks.

Intuitively, the "simplest" strategy is to play a constant action throughout. Anything else requires *attention* [3]: to measure the state and to switch to a different action if necessary. Measuring the state can involve running code on the platform activating sensors and processing sensed values, and using network bandwidth or bus slots to transmit the sensed values to a central processor. Switching to a different action may require dynamic computation of lower-level control laws implementing these actions, switching modes and tasks, as well as re-scheduling bus or network slots. The more frequently these tasks must be performed, the more attention is required.

Of course, there may not be a winning strategy that plays a constant action throughout. Consider in Figure 1 the objective of visiting t infinitely often. Again, any action is possible from states $\{ 1,\ldots,n \}$, as long as 0 is played at s and 1 at t. A possible strategy can, starting from 1, play 1 for n steps, then 0 for a step, then 1 for $n+1$ steps, etc., or dually, play 0 for $n+1$ steps, 1 for $n+1$ steps, a single 0, etc. Both strategies can be implemented with lower processor requirements than one that alternates between 0 and 1. The precise strategy chosen will depend on the actual costs involved in switching between 0 and 1. In general, the lowest-cost controller must optimize the usage, over the long run, of system resources while ensuring the winning condition. Formally, the controller must optimize costs associated with measuring state and switching controllers while ensuring the winning condition is satisfied.

In this paper, we consider the problem of minimum attention controllers for ω-regular objectives. We introduce a cost for measuring the state, as well as a cost for changing the control action, in the model of two-player games. We then ask for a strategy which ensures the ω-regular objective while minimizing the long-run costs incurred due to measurement and switching actions. Technically, playing a game without measuring the state or changing the control action is similar to playing a game of incomplete information [33,22,7]. We formulate the minimum attention control problem as a game of incomplete information, and we show a polynomial-time reduction from the problem of minimum attention control for ω-regular objectives to solving a mean-payoff parity condition [8] on a game of incomplete information. Together with results on incomplete information games [7], this gives an EXPTIME-complete procedure when the winning

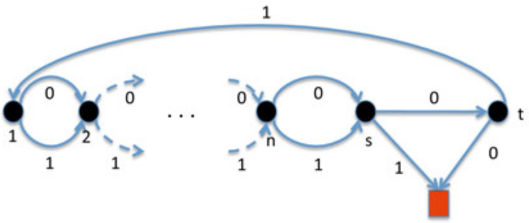


Fig. 1. Simple example

objective is given as a parity condition on states, and a triply exponential procedure when the objective is given in linear-temporal logic.

We develop the theory both for finite-state, discrete control problems, as well as for infinite state systems for which there is a finite bisimulation quotient. Using known results about stable partitions of timed games [1,24] and rectangular automata [16], it follows that the minimum attention controller synthesis problem is decidable for timed games and discrete-time control for rectangular automata.

Attention, and minimum attention controllers, were introduced in a seminal paper by Brockett [3]. There, the problem of minimum attention synthesis is formulated for controlled dynamical systems, and set up as the minimization problem for an attention index, a functional involving the partial derivatives $\frac{\partial u}{\partial t}$ and $\frac{\partial u}{\partial x}$ of the control function u w.r.t. time and state, respectively, subject to constraints on u to guarantee a minimum level of system performance. The resulting problem involves minimization of non-linear functions subject to systems of partial differential equations, even in the case of linear control systems. Minimum attention control was applied to solve control problems for vehicular control [4] and for control under network bandwidth constraints [27], but the algorithmic complexity of the methods are not immediate. More recently, [2] studies approximations of the problem using event-driven control.

Solving the minimum attention control problem for discrete systems suggests a computational approach to *approximately* solve the minimum attention synthesis problem for controlled dynamical system. The link between continuous dynamical systems and discrete systems is provided by *approximate abstractions* of continuous models [30,14]. Approximate abstractions generalize the classical language-theoretic notions of language containment and simulation to the quantitative case; an ε-approximate abstraction of a continuous system is a discrete system such that for any trace of the original system, there is a trace of the abstract system which is at a distance of at most ε, for a design parameter ε. With approximate abstraction relations, the minimum attention control problem for dynamical systems can be approximately solved in two steps: first compute the abstraction, then solve the problem on the discrete abstraction.

Related Works. In this work we consider the minimum attention controller synthesis problem and show that the problem can be solved by solving a special class of incomplete-information mean-payoff parity games and is EXPTIME-complete. The general problem of incomplete-information mean-payoff games was studied in [11] and the problem was shown to be undecidable, whereas we show that the minimum attention synthesis problem belongs to a decidable subclass. The problem of mean-payoff parity games was studied in [8] but in the setting of perfect-information games, and we show that for the special class of incomplete-information games we obtain, the problem is decidable using solutions of [8]. Incomplete-information games with Boolean objectives (such as parity objectives) were considered in [7], whereas in this paper the problem we consider reduces to incomplete-information games with mixed quantitative and Boolean objective (combination of mean-payoff and parity objectives). The problem of fault diagnosis with static and dynamic observers has been considered in [6], and it was shown that the static observer problem is NP-complete and the dynamic observer problem can be solved in 2EXPTIME using solution of mean-payoff games. In contrast our problem requires solution of mean-payoff parity games and is EXPTIME-complete. The fault diagnosis problem was also considered in [36] where a dynamic programming approach was used to solve the problem. No complexity bounds are known. In contrast our approach is game theoretic and we establish optimal complexity bounds.

2 Preliminaries

In this section we present the required preliminaries. We first present the mathematical framework of imperfect information games, and then present a reduction of imperfect information games to perfect information games. In the following section we will use the definitions and the results of this section to develop the theory of minimum attention control.

2.1 Imperfect Information Games

A *game structure* (*of imperfect information*) is a tuple $G = \langle L, l_0, \Sigma, \Delta, \mathcal{O}, \gamma \rangle$, where L is a finite set of states, $l_0 \in L$ is the initial state, Σ is a finite alphabet (of input letters or actions), $\Delta \subseteq L \times \Sigma \times L$ is a set of labeled transitions, $\mathcal{O}$ is a finite set of observations, and $\gamma : \mathcal{O} \to 2^L \backslash \emptyset$ maps each observation to the set of states that it represents. We require the following two properties on G: (i) for all $\ell \in L$ and all $\sigma \in \Sigma$, there exists $\ell' \in L$ such that $(\ell, \sigma, \ell') \in \Delta$; and (ii) the set $\{\gamma(o) \mid o \in \mathcal{O}\}$ partitions L. We say that G is a game structure of *perfect information* if $\mathcal{O} = L$ and $\gamma(\ell) = \{\ell\}$ for all $\ell \in L$. We omit $(\mathcal{O}, \gamma)$ in the description of games of perfect information. For $\sigma \in \Sigma$ and $s \subseteq L$, let $\mathsf{Post}^G_\sigma(s) = \{\ell' \in L \mid \exists \ell \in s : (\ell, \sigma, \ell') \in \Delta\}$.

In a game structure, in each turn, Player 1 (controller) chooses a letter in Σ, and Player 2 (system or plant) resolves nondeterminism by choosing the successor state. A *play* in G is an infinite sequence $\pi = \ell_0 \sigma_0 \ell_1 \ldots \sigma_{n-1} \ell_n \sigma_n \ldots$ such that (i) $\ell_0 = l_0$, and (ii) for all $i \geq 0$, we have $(\ell_i, \sigma_i, \ell_{i+1}) \in \Delta$. The *prefix up to* ℓ_n of the play π is denoted by $\pi(n)$; its *length* is $|\pi(n)| = n + 1$; and its *last element* is $\mathsf{Last}(\pi(n)) = \ell_n$. The *observation sequence* of π is the unique infinite sequence $\gamma^{-1}(\pi) = o_0 \sigma_0 o_1 \ldots \sigma_{n-1} o_n \sigma_n \ldots$ such that for all $i \geq 0$, we have $\ell_i \in \gamma(o_i)$. Similarly, the *observation sequence* of $\pi(n)$ is the prefix up to o_n of $\gamma^{-1}(\pi)$. The set of infinite plays in G is denoted $\mathsf{Plays}(G)$, and the set of corresponding finite prefixes is denoted $\mathsf{Prefs}(G)$. A state $\ell \in L$ is *reachable* in G if there exists a prefix $\rho \in \mathsf{Prefs}(G)$ such that $\mathsf{Last}(\rho) = \ell$. The *knowledge* associated with a finite observation sequence $\tau = o_0 \sigma_0 o_1 \sigma_1 \ldots \sigma_{n-1} o_n$ is the set $\mathsf{K}(\tau)$ of states in which a play can be after this sequence of observations, that is, $\mathsf{K}(\tau) = \{\mathsf{Last}(\rho) \mid \rho \in \mathsf{Prefs}(G) \text{ and } \gamma^{-1}(\rho) = \tau\}$.

The following lemma presents a inductive construction of the knowledge. The proof of the lemma is standard.

Lemma 1. *Let* $G = \langle L, l_0, \Sigma, \Delta, \mathcal{O}, \gamma \rangle$ *be a game structure. For* $\sigma \in \Sigma$, $\ell \in L$, *and* $\rho, \rho' \in \mathsf{Prefs}(G)$ *with* $\rho' = \rho \cdot \sigma \cdot \ell$, *let* $o_\ell \in \mathcal{O}$ *be the unique observation such that* $\ell \in \gamma(o_\ell)$. *Then* $\mathsf{K}(\gamma^{-1}(\rho')) = \mathsf{Post}^G_\sigma(\mathsf{K}(\gamma^{-1}(\rho))) \cap \gamma(o_\ell)$.

Strategies. A *strategy* in G for Player 1 is a function $\alpha : \mathsf{Prefs}(G) \to \Sigma$ that given a finite prefix or history of a play specifies the next input letter or action. A strategy α for Player 1 is *observation-based* if for all prefixes $\rho, \rho' \in \mathsf{Prefs}(G)$, if $\gamma^{-1}(\rho) = \gamma^{-1}(\rho')$, then $\alpha(\rho) = \alpha(\rho')$. In games of imperfect information we are interested in the existence of observation-based strategies for Player 1. A *strategy* in G for Player 2 is a function $\beta : \mathsf{Prefs}(G) \times \Sigma \to L$ such that for all $\rho \in \mathsf{Prefs}(G)$ and all $\sigma \in \Sigma$, we have $(\mathsf{Last}(\rho), \sigma, \beta(\rho, \sigma)) \in \Delta$. We denote by $\mathcal{A}_G$, $\mathcal{A}^O_G$, and $\mathcal{B}_G$ the set of all Player-1 strategies, the set of all observation-based Player-1 strategies, and the set of all Player-2 strategies in G, respectively.

The *outcome* of two strategies α (for Player 1) and β (for Player 2) in G is the play $\pi = \ell_0\sigma_0\ell_1 \ldots \sigma_{n-1}\ell_n\sigma_n \ldots \in \mathsf{Plays}(G)$ such that for all $i \geq 0$, we have $\sigma_i = \alpha(\pi(i))$ and $\ell_{i+1} = \beta(\pi(i), \sigma_i)$. This play is denoted $\mathsf{outcome}(G, \alpha, \beta)$. The *outcome* of a strategy α for Player 1 in G is the set $\mathsf{Outcome}_1(G, \alpha)$ of plays π such that there exists a strategy β for Player 2 with $\pi = \mathsf{outcome}(G, \alpha, \beta)$. The outcome sets for Player 2 are defined symmetrically.

Qualitative objectives. A *qualitative objective* for G is a set ϕ of infinite sequences of states and input letters, that is, $\phi \subseteq (L \times \Sigma)^\omega$. A play $\pi = \ell_0\sigma_0\ell_1 \ldots \sigma_{n-1}\ell_n\sigma_n \ldots \in \mathsf{Plays}(G)$ *satisfies* the objective ϕ, denoted $\pi \models \phi$, if $\pi \in \phi$. We assume objectives are Borel measurable, that is, a qualitative objective is a Borel set in the Cantor topology on $(L \times \Sigma)^\omega$ [18].

We specifically consider *parity objectives* [12,35]. Parity objectives are a canonical form to express all ω-regular objectives [35] and lie in the intersection $\Sigma_3 \cap \Pi_3$ of the third levels of the Borel hierarchy. For a play $\pi = \ell_0\sigma_0\ell_1 \ldots$, we write $\mathsf{Inf}(\pi)$ for the set of states that appear infinitely often in π, that is, $\mathsf{Inf}(\pi) = \{\ell \in L \mid \ell_i = \ell \text{ for infinitely many } i\text{'s}\}$. For $d \in \mathbb{N}$, let $p : L \to \{\, 0, 1, \ldots, d \,\}$ be a *priority function*, which maps each state to a nonnegative integer priority. The *parity* objective $\mathsf{Parity}(p)$ requires that the minimum priority that appears infinitely often be even. Formally, $\mathsf{Parity}(p) = \{\, \pi \mid \min\{\, p(\ell) \mid \ell \in \mathsf{Inf}(\pi) \,\} \text{ is even} \,\}$. Observe that the objectives are defined on sequence of state and input letters, and not on observation and input letters.

Quantitative objectives. In addition to parity (ω-regular) objectives, our algorithms will require solving games with quantitative objectives. A *quantitative objective* for G is a Borel measurable function f on infinite sequences of observations and input letters to reals, that is, $f : (L \times \Sigma)^\omega \to \mathbb{R} \cup \{\, \infty, -\infty \,\}$. We specifically consider mean-payoff and mean-payoff parity objectives. Let $r : \Sigma \to \mathbb{R}$ be a reward-function that maps every input letter σ to a real-valued reward $r(\sigma)$, and let $p : L \to \{\, 0, 1, \ldots, d \,\}$ be a priority function. We define the mean-payoff and mean-payoff parity objectives as follows.

1. *Mean-payoff objectives.* For a play $\pi = \ell_0\sigma_0\ell_1 \dots \sigma_{n-1}\ell_n\sigma_n \dots$ the mean-payoff objective is the long-run average of the rewards of the input letters [38]. Formally, for a reward function $r : \Sigma \to \mathbb{R}$, the mean-payoff objective is a function $\mathsf{M}(r)$ from plays to reals that maps the play $\pi = \ell_0\sigma_0\ell_1 \dots \sigma_{n-1}\ell_n\sigma_n \dots$ to $\mathsf{M}(r)(\pi) = \limsup_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} r(\sigma_i)$.
2. *Mean-payoff parity objectives.* For a play $\pi = \ell_0\sigma_0\ell_1 \dots \sigma_{n-1}\ell_n\sigma_n \dots$ the mean-payoff parity objective is the long-run average of the rewards of the input letters if the parity objective is satisfied and $-\infty$ otherwise. Formally, for a reward function $r : \Sigma \to \mathbb{R}$ and a priority function p, the mean-payoff parity objective is a function $\mathsf{MP}(p,r)$ defined on plays as follows: for a play $\pi = \ell_0\sigma_0\ell_1 \dots \sigma_{n-1}\ell_n\sigma_n \dots$ we have $\mathsf{MP}(p,r)(\pi) = \mathsf{M}(\pi)$ if $\pi \in \mathsf{Parity}(p)$, and $\mathsf{MP}(p,r)(\pi) = -\infty$ otherwise.

Observe that the reward function are on input letters, rather than transition of the game graph. If we consider reward function on transitions, then mean-payoff games with imperfect information is undecidable [11], whereas if the rewards are on input letters, then the problem is EXPTIME-complete (Corollary 1).

Sure winning and optimal winning. A strategy λ_i for Player i in G is *sure winning* for a qualitative objective ϕ if for all $\pi \in \mathsf{Outcome}_i(G, \lambda_i)$, we have $\pi \models \phi$. A strategy λ_i for Player i in G is *optimal* for a quantitative objective f if for all strategies λ for Player i we have $\inf_{\pi\in\mathsf{Outcome}_i(G,\lambda_i)} f(\pi) \geq \inf_{\pi\in\mathsf{Outcome}_i(G,\lambda)} f(\pi)$. The following theorem from Martin [25] states that perfect-information games with (qualitative or quantitative) Borel objectives are *determined*: from each state, either Player 1 or Player 2 wins (for qualitative objectives), or a value can be defined (for quantitative objectives).

Theorem 1 (Determinacy). [25] *(1) For all perfect-information game structures G and all qualitative Borel objectives ϕ, either there exists a sure-winning strategy for Player 1 for the objective ϕ, or there exists a sure-winning strategy for Player 2 for the complementary objective* $\mathsf{Plays}(G) \setminus \phi$. *(2) For all perfect-information game structures G and all quantitative Borel objectives f, we have* $\sup_{\alpha\in\mathcal{A}} \inf_{\pi\in\mathsf{Outcome}(G,\alpha)} f(\pi) = \inf_{\beta\in\mathcal{B}} \sup_{\pi\in\mathsf{Outcome}(G,\beta)} f(\pi)$.

2.2 From Imperfect-Information to Perfect-Information

In this subsection we present results related to reduction of imperfect information games to perfect information games by subset construction. First, we use the results of [7] to show that a game structure G of imperfect information can be encoded by a game structure G^{K} of perfect information such that for every qualitative Borel objective ϕ, there is an observation-based sure-winning strategy for Player 1 in G for ϕ if and only if there is a sure-winning strategy for Player 1 in G^{K} for ϕ. The same construction works for quantitative Borel objectives. We obtain G^{K} using a subset construction. Each state in G^{K} is a set of states of G representing the knowledge of Player 1. In the worst case, the size of G^{K} is exponentially larger than the size of G.

Subset construction. Given a game structure of imperfect information $G = \langle L, l_0, \Sigma, \Delta, \mathcal{O}, \gamma\rangle$, we define the *knowledge-based subset construction* of G as the following game structure of perfect information: $G^{\mathsf{K}} = \langle \mathcal{L}, \{l_0\}, \Sigma, \Delta^{\mathsf{K}}\rangle$, where

$\mathcal{L} = 2^L \backslash \{\emptyset\}$, and $(s_1, \sigma, s_2) \in \Delta^{\mathsf{K}}$ iff there exists an observation $o \in \mathcal{O}$ such that $s_2 = \mathsf{Post}^G_\sigma(s_1) \cap \gamma(o)$ and $s_2 \neq \emptyset$. Notice that for all $s \in \mathcal{L}$ and all $\sigma \in \Sigma$, there exists a set $s' \in \mathcal{L}$ such that $(s, \sigma, s') \in \Delta^{\mathsf{K}}$. Given a game structure of imperfect information G we refer to the game structure G^{K} as $\mathsf{Pft}(G)$.

Lemma 2 ([7]). *For all sets $s \in \mathcal{L}$ that are reachable in G^{K}, and all observations $o \in \mathcal{O}$, either $s \subseteq \gamma(o)$ or $s \cap \gamma(o) = \emptyset$.*

By an abuse of notation, we define the *observation sequence* of a play $\pi = s_0\sigma_0 s_1 \ldots \sigma_{n-1} s_n \sigma_n \ldots \in \mathsf{Plays}(G^{\mathsf{K}})$ as the infinite sequence $\gamma^{-1}(\pi) = o_0\sigma_0 o_1 \ldots \sigma_{n-1} o_n \sigma_n \ldots$ of observations such that for all $i \geq 0$, we have $s_i \subseteq \gamma(o_i)$. Since the observations partition the states, and by Lemma 2, this sequence is unique. The play π *satisfies* an objective $\phi \subseteq (\mathcal{O} \times \Sigma)^\omega$ if $\gamma^{-1}(\pi) \in \phi$. As above, we say that a play $\pi = s_0\sigma_0 s_1 \ldots \sigma_{n-1} s_n \sigma_n \cdots \in \mathsf{Plays}(G^{\mathsf{K}})$ *satisfies* an objective ϕ iff the sequence of observations $o_0 o_1 \ldots o_n \ldots$ such that for all $i \geq 0$, $\ell_i \in \gamma(o_i)$ belongs to ϕ. The following lemma follows from the results of [7].

Lemma 3 ([7]). *If Player 1 has a sure-winning strategy in G^{K} for an objective ϕ, then Player 1 has an observation-based sure-winning strategy in G for ϕ. If Player 1 does not have a deterministic sure-winning strategy in G^{K} for a Borel objective ϕ, then Player 1 does not have an observation-based sure-winning strategy in G for ϕ.*

Together with Theorem 1, Lemma 3 implies the first part of the following theorem, also used in [7]. The second part of the theorem generalizes the result to quantitative Borel objectives. The proofs can be found in [7,9].

Theorem 2. *Let G be a game structure, and $G^{\mathsf{K}} = \mathsf{Pft}(G)$. The following assertions hold. (1) Player 1 has an observation-based sure-winning strategy in G for a qualitative Borel objective ϕ if and only if Player 1 has a sure-winning strategy in G^{K} for ϕ. (2) $\sup_{\alpha \in \mathcal{A}^O_G} \inf_{\pi \in \mathsf{Outcome}(G,\alpha)} f(\pi) = \sup_{\alpha \in \mathcal{A}_{G^{\mathsf{K}}}} \inf_{\pi \in \mathsf{Outcome}(G,\alpha)} f(\pi)$.*

Theorem 2 and the results of [8] on perfect information mean-payoff parity games show that imperfect information mean-payoff parity games can be solved in EXPTIME, and an EXPTIME lower bound follows from the lower bound for imperfect information parity games [7]. We have the following corollary.

Corollary 1. *Given an imperfect information game structure G, a priority function $p : L \to \{\, 0, 1, \ldots, d \,\}$ and a reward function $r : \Sigma \cdot \mathbb{R}$, the decision problem of whether $\sup_{\alpha \in \mathcal{A}^O_G} \inf_{\pi \in \mathsf{Outcome}(G,\alpha)} \mathsf{MP}(p, r)(\pi) \geq \nu$, for a rational threshold ν, is EXPTIME-complete.*

3 Minimum Attention Control

We now consider minimum attention control of imperfect information games. We will present a polynomial reduction of the minimum attention control problem for imperfect information games to the classical imperfect information games presented in the previous section. We will also show that the minimum attention control problem is EXPTIME-complete.

Switching and Monitoring Costs. We associate two kinds of costs for control: *switching costs* and *monitoring costs*. The switching cost is incurred when the control switches between two input letters, and the monitoring cost is incurred when the controller monitors the state of the plant (i.e., the current observation). Formally, let $\mathsf{cost} : \Sigma \times \Sigma \to \mathbb{R}$ denote the cost of switching between two input letters (or actions), i.e., $\mathsf{cost}(\sigma, \sigma')$ denote the cost of switching from input letter σ to input letter σ'. Let mon denote the cost of monitoring, i.e., monitoring the current observation of the plant. Given a ω-regular specification specified as a parity objective, the goal of the controller is to ensure the parity objective minimizing the long-run average cost of switching and monitoring. We now formally present monitor-action strategies, the notion of cost of a play, then the notion of minimum attention control, and finally the reduction of minimum attention control problem to imperfect information games with mean-payoff parity objective.

Monitor-Action Strategies. Let $G = \langle L, l_0, \Sigma, \Delta, \mathcal{O}, \gamma \rangle$ be a game structure of imperfect information. Let $\widehat{\Sigma} = \{ 0, 1 \} \times \Sigma$, be the action for controller where the first component denotes monitoring or not (0 denotes no monitoring and 1 denotes monitoring). Let $\widehat{o}$ be a new observation not in $\mathcal{O}$, and let $\widehat{\gamma}$ be a new observation mapping such that $\widehat{\gamma}(o) = \gamma(o)$ for $o \in \mathcal{O}$, and $\widehat{\gamma}(\widehat{o}) = L$. If player 1 chooses not to monitor, then player 1 does not see the current observation (this is equivalent to say that player 1 gets to observe $\widehat{o}$). A *monitor-action* strategy for player 1 is a function $\widehat{\alpha} : (L \times \widehat{\Sigma})^* \times L \to \widehat{\Sigma}$. Given a play $\pi = l_0 \widehat{\sigma}_0 l_1 \widehat{\sigma}_1 l_2 \widehat{\sigma}_2 \ldots$, the observation sequence $\widehat{\gamma}^{-1}\pi = o_0 \widehat{\sigma}_0 o_1 \widehat{\sigma}_1 o_2 \widehat{\sigma}_2 \ldots$, where $o_i = \gamma^{-1}(l_i)$ if $\sigma_i \in \{ 1 \} \times \Sigma$, and $\widehat{o}$ otherwise. A monitor-action strategy $\widehat{\alpha}$ is observation-based, if for all finite prefixes $\widehat{\rho}, \widehat{\rho'} \in (L \times \widehat{\Sigma})^* \times L$ such that $\widehat{\gamma}^{-1}(\widehat{\rho}) = \widehat{\gamma}^{-1}(\widehat{\rho'})$ we have $\widehat{\alpha}^{-1}(\widehat{\rho}) = \widehat{\alpha}^{-1}(\widehat{\rho'})$.

Cost of a Play. Given a play $\pi = l_0 \widehat{\sigma}_0 l_1 \widehat{\sigma}_1 l_2 \widehat{\sigma}_2 \ldots$, the monitor-switching cost of π is as follows. For $i > 1$ and $z \in \{ 0, 1 \}$, let $\widehat{c}(\widehat{\sigma}_i) = \mathsf{cost}(\sigma_{i-1}, \sigma_i)$ if $\widehat{\sigma}_i = (0, \sigma_{i-1})$ and $\widehat{\sigma}_{i-1} = (z, \sigma_{i-1})$, and $\widehat{c}(\widehat{\sigma}_i) = \mathsf{cost}(\sigma_{i-1}, \sigma_i) + \mathsf{mon}$ if $\widehat{\sigma}_i = (1, \sigma_{i-1})$ and $\widehat{\sigma}_{i-1} = (z, \sigma_{i-1})$. denote the cost of monitoring and switching in the i-th step. Then the cost of the play is defined as the long-run average of the monitoring and switching cost, i.e., $\widehat{c}(\pi) = \limsup_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} \widehat{c}(\widehat{\sigma}_{i-1}, \widehat{\sigma}_i)$.

Minimum Attention Control. Given an imperfect information game structure G, and a parity objective ϕ, a monitor-action strategy $\widehat{\alpha}$ is ν-frugal iff the following conditions hold: (a) $\widehat{\alpha}$ is obervation-based; (b) for all π in $\mathsf{Outcome}(G, \widehat{\alpha})$ we have $\pi \in \phi$, i.e., the parity objective is ensured; and (c) $\widehat{c}(\pi) \leq \nu$, i.e., the monitor-switching cost is at most ν. In other words, the strategy $\widehat{\alpha}$ ensures the parity objective without incurring monitor-switching cost more than ν.

Reduction of Games with Move Assignment. We will present a reduction of the minimum attention control problem to imperfect information games with mean-payoff parity objectives. We first consider an extension of imperfect information games where there is an input assignment function $\Gamma_1 : L \to 2^{\Sigma} \setminus \emptyset$, that assigns to every state ℓ the set of available input letters $\Gamma_1(\ell)$, i.e., every input letter may not be available at every state. Imperfect information games with input assignment function can be reduced to imperfect information games with no input assignment function as follows: (1) add an additional absorbing state $\widetilde{\ell}$ that is loosing for player 1; (2) for a state ℓ and an input letter $\sigma \in \Sigma \setminus \Gamma_1(\ell)$ not available at ℓ by the input assignment, we add σ as available input, and add a transition from ℓ to the loosing state $\widetilde{\ell}$ for σ. Thus it is ensured if player 1 chooses an input that is not available, then player 1 loses immediately.

Reduction of Minimum Attention Control. Let $G = \langle L, l_0, \Sigma, \Delta, \mathcal{O}, \gamma \rangle$ be a game structure of imperfect information with a priority function $p : L \to \{ 0, 1, \ldots, d \}$. For minimum attention control we construct a game structure of imperfect information with move assignment as follows: the game structure is $\widetilde{G} = \langle \widetilde{L}, \widetilde{l_0}, \widetilde{\Sigma}, \widetilde{\Delta}, \widetilde{\mathcal{O}}, \widetilde{\gamma}, \widetilde{\Gamma}_1 \rangle$ along with cost function $\widetilde{c} : \widetilde{\Sigma} \to \mathbb{R}$. We describe the components below.

1. *State space.* We have $\widetilde{L} = (L \times \{ 0, 1, 2 \} \times \Sigma) \cup (\{ l_0 \} \times \{ 0, 1, 2 \})$. The first component is the state of G, the second component is 0 or 1 depending on whether player 1 decides to monitor or not, and 2 if it is player 1's turn to decide whether to monitor or not. The third component is an input letter (to remember the choice of last letter of player 1). Additionally, there are states of the form (l_0, j) for $j \in \{ 0, 1, 2 \}$. The starting state $\widetilde{l_0}$ is $(l_0, 2)$.
2. *Input letters.* We have $\widetilde{\Sigma} = (\Sigma \times \Sigma) \cup \{ 0, 1 \}$, i.e., the set of input letters is a pair of input letters of the original game (switching between input letters) with the $\{ 0, 1 \}$ to denote the choice of monitoring.
3. *Observation.* We have $\widetilde{\mathcal{O}} = \mathcal{O} \cup \{ \widetilde{o} \}$, where $\widetilde{o}$ is a new observation.
4. *Move assignment.* We have $\widetilde{\Gamma}_1((\ell, 2, \sigma)) = \{ 0, 1 \}$ for $\ell \in L$ and $\sigma \in \Sigma$; and $\widetilde{\Gamma}_1((\ell, j, \sigma)) = \{ (\sigma, \sigma') \mid \sigma' \in \Sigma \}$ for $\ell \in L, j \in \{ 0, 1 \}$ and $\sigma \in \Sigma$. At states where the second component is 2, player 1 can choose between two input letters: 0 to denote no monitoring, and 1 to denote monitoring. At states where the second component is 0 or 1, player 1 can choose input letters matching with the input letter of the state (player 1 specifies the switching from the last letter to a new letter). Similarly, we have $\Gamma_1((l_0, 2)) = \{ 0, 1 \}$ and $\Gamma_1((l_0, j)) = \Sigma$, for $j \in \{ 0, 1 \}$.
5. *Transition function.* We have the following cases: (a) for states $(\ell, 2, \sigma)$ we have $((\ell, 2, \sigma), j, (\ell, j, \sigma)) \in \widetilde{\Delta}$, for $j \in \{ 0, 1 \}$, i.e., given the choice of input letter only the second component of state changes according to the input letter; (b) for states (ℓ, j, σ) with $j \in \{ 0, 1 \}$ we have $((\ell, j, \sigma), (\sigma, \sigma')(\ell', 2, \sigma')) \in \widetilde{\Delta}$ iff $(\ell, \sigma', \ell') \in \Delta$, i.e., the transition of the game structure is mimicked according to the first component, the second component changes to 2, and the last input letter is remembered in the third component; (c) for state $(l_0, 2)$ we have $((l_0, 2), j, (l_0, j)) \in \widetilde{\Delta}$ for $j \in \{ 0, 1 \}$; and (d) for states (l_0, j), with $j \in \{ 0, 1 \}$ we have $((l_0, j), \sigma'(\ell', 2, \sigma')) \in \widetilde{\Delta}$ iff $(l_0, \sigma', \ell') \in \Delta$.
6. *Observation mapping.* We have (a) $\widetilde{\gamma}^{-1}((\ell, j, \sigma)) = \widetilde{\gamma}^{-1}((l_0, j)) = \widetilde{o}$, for $j \in \{ 0, 2 \}$; and (b) $\widetilde{\gamma}^{-1}((\ell, 1, \sigma)) = \gamma(\ell)$ and $\widetilde{\gamma}^{-1}((l_0, 1)) = \gamma(l_0)$; i.e., when the second component is 0 or 2, then player 1 is not monitoring and hence observes nothing, and otherwise if the second component is 1, then player 1 is monitoring and hence observes the observation of the original game.
7. *Cost function.* We have $\widetilde{c}(0) = 0$ (no cost); $\widetilde{c}(1) = -\mathsf{mon}$ (cost of monitoring); and $\widetilde{c}((\sigma, \sigma')) = -\mathsf{cost}(\sigma, \sigma')$ (cost of switching).
8. *Parity function.* The priority function $\widetilde{p} : \widetilde{L} \to \{0, 1, \ldots, d\}$ is obtained as follows: for a state $\widetilde{\ell} \in \widetilde{L}$ the priority $\widetilde{p}(\widetilde{\ell})$ is $p(\ell)$, where ℓ is the first component of $\widetilde{\ell}$.

There is a one-to-one correspondence between monitor-action strategies that are observation-based in G, and observation-based strategies in the game $\widetilde{G}$. The cost incurred in G in every step is incurred in two steps in $\widetilde{G}$ as in $\widetilde{G}$ we mimic the choice of monitoring and choice of action switch of G in two steps. Hence we have the following lemma.

Lemma 4. *Let G be a game structure of imperfect information, and let $p : L \to \{ 0, 1, \ldots, d \}$ be a priority function. There is a monitor-action strategy $\widehat{\alpha}$ in G that is ν-frugal for the objective* $\mathsf{Parity}(p)$ *iff* $\sup_{\alpha \in \mathcal{A}^O_{\widetilde{G}}} \inf_{\pi \in \mathsf{Outcome}(\widetilde{G}, \alpha)} \mathsf{MP}(\widetilde{c}, \widetilde{p})(\pi) \geq -\frac{\nu}{2}$.

We have the following result for minimum attention control: (a) the EXPTIME upper bound follows from Corollary 1 and Lemma 4 and the fact that our reduction from G to $\widetilde{G}$ is polynomial; (b) the lower bound follows from EXPTIME-hardness of imperfect information parity games: with monitoring and switching costs both set to 0, the minimum attention control problem is the same as winning an imperfect information parity game.

Theorem 3. *The minimum attention control problem for imperfect information game structures with parity objectives is EXPTIME-complete.*

We have assumed that the winning objective is given as a parity condition on the state space of the game. If instead, we are given a two-player game structure, and separately, a specification in linear-temporal logic (LTL) [28], then standard automata-theoretic constructions [34] can be used to reduce the problem to our case. That is, from the LTL specification φ, one constructs a deterministic parity automaton whose size is at most doubly exponential in the size of φ and whose number of parities is exponential in the size of φ. A synchronous product of the game structure with this automaton gives an imperfect information game whose size is the product of the size of the original game and the size of the automaton. The solution to the imperfect information game involves a subset construction, adding an extra exponential, and then solving a mean-payoff parity game which is polynomial in the size of the game and exponential in the number of parities. But a triple exponential raised to a single exponential is still triple exponential, so we conclude the following.

Corollary 2. *The minimum attention control problem for imperfect information game structures and linear-temporal logic specifications is in 3EXPTIME.*

We note that the high complexity of our procedure is disappointing, and unlikely to yield an efficient tool. It will be interesting to see if more efficient algorithms can be designed for fragments of LTL.

4 Infinite State Systems

We now apply the theory of minimum attention controller synthesis to the discrete time control problem for *rectangular automata* [16]. We obtain our results using a general decidability result about imperfect-information games on infinite state spaces that have a stable partition with a finite quotient.

R-stable games. In this section we drop the assumption of finite state space of games. Let $G = \langle L, l_0, \Sigma, \Delta, \mathcal{O}, \gamma \rangle$ be a game structure of imperfect-information such that L is infinite. Let $R = \{ r_1, r_2, \ldots, r_l \}$ be a finite partition of L. A set $Q \subseteq L$ is *R-definable* if $Q = \bigcup_{r \in Z} r$, for some $Z \subseteq R$. The game G is *R-stable* if the following conditions hold for all $\sigma \in \Sigma$: (a) the set $\{ l \in L \mid \exists l' \in L.(l, \sigma, l') \in \Delta \}$ is R-definable; (b) for all $r \in R$, the set $\mathsf{Post}^G_\sigma(r)$ is R-definable; (c) for all $r, r' \in R$, if for some $x \in r$ we have $\mathsf{Post}^G_\sigma(\{ x \}) \cap r' \neq \emptyset$, then for all $x' \in r$ we have $\mathsf{Post}^G_\sigma(\{ x' \}) \cap r' \neq \emptyset$; and (d) for all $o \in \mathcal{O}$, the set $\gamma(o)$ is R-definable.

Lemma 5. *The following assertions hold. (1) Let G be a game structure of imperfect information, and let R be a finite partition of the state space of G such that the game G is R-stable. Then the perfect-information game* $\mathsf{Pft}(G)$ *is* 2^R*-stable. (2) Let* $\overline{G}$ *be a perfect-information game structure with a parity objective with* d*-priorities and a mean-payoff objective with rewards on* Σ *such that the maximal absolute value of the rewards is* W*. If* $\overline{G}$ *is* $\overline{R}$*-stable, for a given finite partition* $\overline{R}$*, then value for the mean-payoff parity objective in* $\overline{G}$ *can be computed in time* $O((|\overline{R}| \cdot W)^{d+5})$.

We present the definition of rectangular automata with imperfect information and then reduce the minimum attention control problem to the problem of game with imperfect information. Using a result of [16] we establish the game of imperfect information is R-stable for a finite set R.

Rectangular constraints. Let $Y = \{ y_1, y_2, \ldots, y_k \}$ be a set of real-valued variables. A *rectangular inequality* over Y is of the form $x_i \sim d$, where d is an integer constant, and $\sim \in \{ \leq, <, \geq, > \}$. A *rectangular predicate* over Y is a conjunction of rectangular inequalities. We denote the set of rectangular predicates over Y as $\mathit{Rect}(Y)$. The rectangular predicate ϕ defines the set of vectors $[\![\phi]\!] = \{ y \in \mathbb{R}^k \mid \phi[Y := y] \text{ is true} \}$. For $1 \leq i \leq k$, let $[\![\phi]\!]_i$ be the projection on variable y_i of the set $[\![\phi]\!]$. A set of the form $[\![\phi]\!]$, where ϕ is a rectangular predicate, is called a rectangle. Given a non-negative integer $m \in \mathbb{N}$, the rectangular predicate ϕ is m-bounded if $|d| \leq m$, for every conjunct $y_i \sim d$ of ϕ. Let us denote by $\mathit{Rect}_m(Y)$ the set of m-bounded rectangular predicates on Y.

Rectangular automata. A *rectangular automaton of imperfect information* H is a tuple $\langle Q, \mathit{Lab}, \mathit{Edg}, Y, \mathit{Init}, \mathit{Inv}, \mathit{Flow}, \mathit{Jump}, \mathcal{O}, \gamma \rangle$ where (a) Q is a finite set of locations; (b) Lab is a finite set of *labels*; (b) $\mathit{Edg} \subseteq Q \times \mathit{Lab} \times Q$ is a finite set of *edges*; (d) $Y = \{ y_1, y_2, \ldots, y_k \}$ is a finite set of *variables*; (e) $\mathit{Init} : Q \to \mathit{Rect}(Y)$ gives the *initial condition* $\mathit{Init}(q)$ of a location q; (f) $\mathit{Inv} : Q \to \mathit{Rect}(Y)$ gives the *invariant condition* $\mathit{Inv}(q)$ of location q (i.e., the automaton can stay in q as long as the values of variables lie in $[\![\mathit{Inv}(v)]\!]$); (g) $\mathit{Flow} : Q \to \mathit{Rect}(\dot{Y})$ governs the evolution of the variables in each location; (h) Jump maps each edge e to a predicate $\mathit{Jump}(e)$ of the form $\phi \wedge \phi' \wedge \bigwedge_{i \not\in \mathit{Update}(e)} (y'_i = y_i)$, where $\phi \in \mathit{Rect}(Y)$, $\phi' \in \mathit{Rect}(Y')$, and $\mathit{Update}(e) \subseteq \{1, 2, \ldots, k\}$; (i) $\mathcal{O}$ is a finite set of observations and $\gamma : \mathcal{O} \to 2^Q \setminus \emptyset$ is the observation mapping such that $\{ \gamma(o) \mid o \in \mathcal{O} \}$ is a partition of Q. The variables in Y' refer to the updated values of the variables after the edge has been traversed. Each variable y_i with $i \in \mathit{Update}(e)$ is updated nondeterministically to a new value in $[\![\phi']\!]_i$. A rectangular automaton is m-bounded if all rectangular constraints are m-bounded. A rectangular automaton is called a *timed automaton* if for each variable $y \in Y$ and each state $q \in Q$, we have $1 \leq \mathit{Flow}(q)(\dot{y}) \leq 1$.

Nondecreasing and bounded variables. Let H be a rectangular automaton, and let $i \in \{ 1, 2, \ldots, k \}$. The variable y_i of H is *nondecreasing* if for all $q \in Q$, the invariant interval $[\![Inv(q)]\!]_i$ and the flow interval $[\![Flow(q)]\!]_i$ are subsets of the nonnegative reals. The variable y_i of H is *bounded* if for all $q \in Q$, the invariant interval $[\![Inv(q)]\!]_i$ is a bounded set. The automaton H is bounded (resp. nondecreasing) if all the variables are bounded (resp. nondecreasing). In sequel we consider automata that are bounded or nondecreasing.

Game semantics. The rectangular automaton game with imperfect information is played as follows: the game starts at a location q and values for the continuous variables $y \in [\![Init(q)]\!]$. At each round the controller can choose to observe (monitor) the observation (paying the monitoring cost) or not; and then the controller decides to take one of the enabled edges (if one exists). Then the environment nondeterministically updates the continuous variables according to the flow predicates by letting time pass for 1 time unit. Then the new round of the game starts. We now present a reduction to imperfect-information game, and then show that the game is stable with respect to a finite partition.

Reduction. A rectangular automaton H with imperfect information $\langle Q, Lab, Edg, Y, Init, Inv, Flow, Jump, \mathcal{O}, \gamma \rangle$ reduces to an infinite state imperfect-information game $\overline{H} = \langle L, l_0, \Sigma, \Delta, \mathcal{O}, \overline{\gamma} \rangle$ as follows:

1. *States.* The set of states is $L = Q \times \mathbb{R}^k$; that is the set of states consists of a tuple of location and values of variables.
2. *Input letters.* The set of input letters is $\Sigma = Lab \cup \{ 1 \}$. The set of input letters is the set of labels Lab of H, and unit time 1.
3. *Observation map.* The observation map is as follows: $\overline{\gamma}(o) = \{ (q, y) \in L \mid \gamma(q) = o \}$.
4. *Transition function.* The transition function is as follows: (a) $((q, y), \sigma, (\overline{q}', y')) \in \Delta$, such that there exists $e = (q, \sigma, q') \in Edg$ with $(y, y') \in [\![Jump(e)]\!]$; and (c) $((q, y), 1, (q, y')) \in \Delta$ such that there exists a continuously differentiable function $f : [0, 1] \to Inv(q)$ such that $f(0) = y$, $f(1) = y'$ and for all $t \in (0, 1)$ we have $\dot{f}(t) \in [\![Flow(q)]\!]$.

The set of observation-based strategies of $\overline{H}$ represents the observation-based strategies for the rectangular automaton game defined by H.

Equivalence relation. Let H be a m-bounded rectangular automaton with imperfect information, and let $\overline{H}$ be the game of imperfect information obtained by the reduction. We define the equivalence relation $\equiv_m$ on the state space as follows: $(q, y) \equiv_m (q', y')$ iff (a) $q = q'$; and (b) for all $1 \leq i \leq k$, either $\lfloor y_i \rfloor = \lfloor y'_i \rfloor$ and $\lceil y_i \rceil = \lceil y'_i \rceil$, or both y_i and y'_i are greater than m. We denote by $R_{\equiv_m}$ the set of equivalence classes of $\equiv_m$. It is easy to observe that $R_{\equiv_m}$ is finite (in fact exponential in the size of H). An extension of the result of [16] gives us the following result.

Lemma 6. *Let H be a m-bounded rectangular automaton game with imperfect information. The imperfect-information game $\overline{H}$ is $R_{\equiv_m}$-stable.*

Theorem 4. *Let H be a rectangular automaton with imperfect information and let $p : Q \to \{ 0, 1, \ldots, d \}$ be a priority function. Let $\overline{p} : Q \cdot \mathbb{R}^k \to \{ 0, 1, \ldots, d \}$ be such that $\overline{p}(q, y) = p(q)$, for $q \in Q$ and $y \in \mathbb{R}^k$. The answer to the ν-frugal problem for H for* $\mathsf{Parity}(p)$ *is true iff the answer to the ν-frugal problem is true in $\overline{H}$ for* $\mathsf{Parity}(\overline{p})$.

From Lemma 5, Lemma 6, Theorem 4, and Theorem 3 we obtain the following corollary.

Corollary 3. *Let H be a rectangular automaton with imperfect information and let $p : Q \to \{ 0, 1, \ldots, d \}$ be a priority function. Whether there is a ν-frugal strategy for the controller in H to satisfy the objective* $\mathsf{Parity}(p)$ *can be decided in 2EXPTIME.*

The result for timed automata follows similarly, using the finite bisimilarity relation for timed automata [1].

5 Discussions

We have presented algorithms for minimum attention controller synthesis for ω-regular objectives purely in the discrete setting, or in a setting (rectangular automata) which can be reduced to the discrete setting. A natural next step is to extend the results in the presence of more general continuous dynamics. One potential direction is to combine the optimization problems in [3] with mean-payoff parity games as described here. This seems hard algorithmically, because the optimization problem in [3] is already quite difficult (and even in the case of linear dynamics, it is not obvious if closed form solutions can be obtained).

A different direction is motivated by work in *approximate abstraction* of continuous control models [30,14,13]. The results in these papers provide techniques to abstract the continuous state space and dynamics of systems to discrete systems such that, any controller for the discrete system is guaranteed to ensure the property in the continuous system up to an error of ϵ, where ϵ is a parameter of the abstraction. Therefore, the study of minimum attention control for hybrid dynamics can be broken into two parts: first, construct a discrete abstraction of the continuous system, and second, apply the techniques described in this paper to solve the problem of minimum attention control.

References

1. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126(2), 183–236 (1994)
2. Anta, A., Tabuada, P.: On the minimum attention and anytime attention problems for nonlinear systems. In: CDC 2010. IEEE, Los Alamitos (2010)
3. Brockett, R.W.: Minimum attention control. In: CDC 1997. IEEE, Los Alamitos (1997)
4. Brockett, R.W.: Minimizing attention in a motion control context. In: CDC 2003. IEEE, Los Alamitos (2003)
5. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Transactions of the AMS 138, 295–311 (1969)
6. Cassez, F., Tripakis, S.: Fault diagnosis with static and dynamic observers. Fundam. Inform. 88(4), 497–540 (2008)

7. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.F.: Algorithms for omega-regular games with imperfect information. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 287–302. Springer, Heidelberg (2006)
8. Chatterjee, K., Henzinger, T.A., Jurdziński, M.: Mean-payoff parity games. In: LICS 2005, pp. 178–187. IEEE, Los Alamitos (2005)
9. Chatterjee, K., Majumdar, R., Henzinger, T.A.: Controller synthesis with budget constraints. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 72–86. Springer, Heidelberg (2008)
10. Church, A.: Logic, arithmetic, and automata. In: Proceedings of the International Congress of Mathematicians, pp. 23–35. Institut Mittag-Leffler (1962)
11. Degorre, A., Doyen, L., Gentilini, R., Raskin, J.-F., Toruńczyk, S.: Energy and mean-payoff games with imperfect information. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 260–274. Springer, Heidelberg (2010)
12. Emerson, E.A., Jutla, C.: Tree automata, mu-calculus and determinacy. In: FOCS 1991, pp. 368–377. IEEE, Los Alamitos (1991)
13. Girard, A.: Synthesis using approximately bisimilar abstractions: state-feedback controllers for safety specifications. In: HSCC, pp. 111–120. ACM, New York (2010)
14. Girard, A., Julius, A.A., Pappas, G.J.: Approximate simulation relations for hybrid systems. Discrete Event Dynamic Systems 18(2), 163–179 (2008)
15. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: STOC 1982, pp. 60–65. ACM, New York (1982)
16. Henzinger, T.A., Kopke, P.W.: Discrete-time control for rectangular hybrid automata. Theoretical Computer Science 221, 369–392 (1999)
17. Jobstmann, B.: Applications and Optimizations for LTL Synthesis. PhD thesis, Graz University of Technology (March 2007)
18. Kechris, A.: Classical Descriptive Set Theory. Springer, Heidelberg (1995)
19. Kloetzer, M., Belta, C.: A fully automated framework for control of linear systems from temporal logic specifications. IEEE Transactions on Automatic Control 53(1), 287–297 (2008)
20. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal logic-based reactive mission and motion planning. IEEE Transactions on Robotics 25(6), 1370–1381 (2009)
21. Kupferman, O., Vardi, M.Y.: μ-calculus synthesis. In: Nielsen, M., Rovan, B. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 497–507. Springer, Heidelberg (2000)
22. Kupferman, O., Vardi, M.Y.: Synthesis with incomplete informatio. In: Advances in Temporal Logic, pp. 109–127. Kluwer Academic Publishers, Dordrecht (2000)
23. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: FOCS 2005: Foundations of Computer Science, pp. 531–540 (2005)
24. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)
25. Martin, D.A.: Borel determinacy. Annals of Mathematics 102(2), 363–371 (1975)
26. Mazo, M., Davitian, A., Tabuada, P.: Pessoa: A tool for embedded controller synthesis. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 566–569. Springer, Heidelberg (2010)
27. Montestruque, L., Antsaklis, P.: On the model-based control of networked systems. Automatica 39(10), 1837–1843 (2003)
28. Pnueli, A.: The temporal logic of programs. In: FOCS 1977, pp. 46–57 (1977)
29. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL 1989, pp. 179–190. ACM, New York (1989)
30. Pola, G., Girard, A., Tabuada, P.: Approximately bisimilar symbolic models for nonlinear control systems. Automatica 44(10), 2508–2516 (2008)

31. Rabin, M.O.: Automata on Infinite Objects and Church's Problem. Conference Series in Mathematics, vol. 13. American Mathematical Society, Providence (1969)
32. Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. IEEE Transactions on Control Theory 77, 81–98 (1989)
33. Reif, J.H.: The complexity of two-player games of incomplete information. Journal of Computer and System Sciences 29, 274–301 (1984)
34. Safra, S.: On the complexity of ω-automata. In: Proceedings of the 29th Annual Symposium on Foundations of Computer Science, pp. 319–327. IEEE Computer Society Press, Los Alamitos (1988)
35. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages. Beyond Words. ch.7, vol. 3, pp. 389–455. Springer, Heidelberg (1997)
36. Thorsley, D., Teneketzis, D.: Active acquisition of information for diagnosis and supervisory control of discrete event systems. Discrete Event Dynamic Systems 17, 531–583 (2007)
37. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science 200(1-2), 135–183 (1998)
38. Zwick, U., Paterson, M.S.: The complexity of mean payoff games on graphs. Theoretical Computer Science 158, 343–359 (1996)

Crossing the Bridge between Similar Games[*]

Jan-David Quesel, Martin Fränzle, and Werner Damm

University of Oldenburg, Department of Computing Science, Germany
{quesel,fraenzle,damm}@informatik.uni-oldenburg.de

Abstract. Specifications and implementations of complex physical systems tend to differ as low level effects such as sampling are often ignored when high level models are created. Thus, the low level models are often not exact refinements of the high level specification. However, they are similar to those. To bridge the gap between those models, we study robust simulation relations for hybrid systems. We identify a family of robust simulation relations that allow for certain bounded deviations in the behavior of a system specification and its implementation in both values of the system variables and timings. We show that for this relaxed version of simulation a broad class of logical properties is preserved. The question whether two systems are in simulation relation can be reduced to a reach avoid problem for hybrid games. We provide a sufficient condition under which a winning strategy for these games exists.

Keywords: formal verification of hybrid systems, robust simulation, logics for hybrid systems, hybrid games.

1 Introduction

Hybrid systems provide a mathematical model for systems with interacting discrete and continuous dynamics. Examples include controllers for trains, cars, and airplanes. In many cases, these controllers are critical for the system operation and safety. Thus, verification is a crucial task during the design of those systems. Unfortunately, verification approaches do not scale well enough to directly tackle the whole implementation of such controllers.

Different approaches have been taken to overcome this issue so far. Refinement between a relaxed model and a synthesized discrete implementation is established by Stauner [21]. His method relies on models that obey certain restrictions. For example the invariants and guards must be overlapping in a single point. Stauner then constructs a sampled implementation that is a refinement of a relaxed version of the model where the guards and invariants overlap in a larger region. The gap from timed systems to implementations was bridged by De Wulf et.al. [25] in a similar way. They provide a relaxed semantics for timed automata that ensures implementability and preservation of LTL properties. Girard et. al. [12] presented an approach how to construct approximately bisimilar symbolic models for switched dynamical systems.

[*] This work was partially supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 160–176, 2011.

Thrane et. al. [22] studied different types of simulations that allow for some deviations of the costs in weighted timed automaton. A notion of simulation in the presence of spatial deviations that allows to related hybrid systems with identical control graphs is presented by Girard et. al. [11].

As effects like sampling that often occur in implementations influence the timing behavior similarity notions have been studied that allow for some deviation in the timing behavior as well. Davoren [5] presented an approach generalizing Skorokhod-metrics and provided conditions under which these initially pseudo-metrics induce topologies which are Hausdorff and are thus indeed metrics. However, she does not present a constructive method for determining the values assigned to two concrete system by the metric.

Inspired by the simulation notion presented by Girard et. al. [11] and the more general similarity notion in Davoren's work [5], (a) we study a simulation relation that allows one system to simulate another in a robust way, where deviations in continuous variable valuations, and in timing behavior are subject to constant bounds, and (b) we then investigate properties preserved under such a simulation relation. Additionally, we drop the requirement inherent to the work in [11] that the discrete behavior must have the same control graph.

We model hybrid systems using a notion of hybrid automata [13]. However, the formalism allows for some specifications that lack real-world realizability. As we want to bridge the gap between specifications and implementations, we do neither consider runs of a system that are Zeno, i.e. an infinite number of transitions is taken within a finite amount of time, nor behaviors that are time blocking. Time blocking means that at a certain point in time, no future evolution of the system is possible due to an invariant preventing any continuous evolution and no transition guard being satisfied. Also, as valuations of transient intermediate variables in internal calculations are not observable, we consider systems similar as long as their outputs are similar, even if transient intermediate values differ.

To determine whether two systems are related by our simulation relation, we build up a two player hybrid game where the existence of a winning strategy for the second player coincides with the fact that the systems are in simulation relation. Hybrid games are a natural extension of timed games [18] where derivatives are no longer restricted to 1 and resets can exhibit a more complex structure. Different types of hybrid games have been studied in the literature so far. Restricted classes of hybrid games have been proven to be decidable (see e.g. [14,2,24]). Unfortunately, the games expressive enough for our purpose do not fall into such a class. Tomlin et. al. [23] study hybrid games for controller synthesis. They give an algorithm how to compute controllable predecessors and thus checking if there is a controller that drives the system into a safe state.

To express properties of real-time as well as hybrid systems a variety of different logics have been proposed (see [1] for a survey). We choose to use a variant of the future fragment of MTL [16] to specify properties of our systems. As basic propositions we use expressions that are evaluated on the system variables. This gives us a nice partitioning between temporal and spatial propositions on a syntactical level which we will exploit when proving that certain properties are preserved by our simulation relation. Henzinger et. al. [15] studied the relation between a timed variant of CTL and a notion of simulation that allows for some bounded deviation in the timing behavior of the timed automaton under consideration. They prove that a certain modification function applied to the temporal operators of their logic ensures that properties can be transfered using this simulation relation. We will use a similar transformation but add some modifications to the basic propositions as well to capture the deviations in space.

With Fainekos and Pappas [7] and Donzé and Maler [6] we share the goal of defining robust satisfaction of linear-time metric temporal logic formulas in a form permitting the generalization of findings obtained on one trajectory to "close-by" trajectories, where "close-by" refers to both space—as already addressed by Fainekos and Pappas—and time—a combination also covered by Donzé and Maler. Our work, however, is notably different from theirs in that they deal with instance properties while we deal with ensemble properties: while theirs robustly evaluates a given formula over a given (sampled) trajectory, we are concerned with computing a simulation relating *every* trajectory of a given concrete system to a robustly corresponding abstract counterpart such that the robust semantics of *every* temporal-logic formula is preserved up to a given tolerance. The latter constitutes a natural notion of system refinement which can help bridge the gap between the abstract models used in system verification and their actual implementations, while the former is the adequate setting for assessing observed trajectories against a given set of requirements.

Contributions. We study a similarity of hybrid systems. In contrast to many other approaches, we do not restrict ourselves to classical refinement, but develop a notion of similarity that relates systems based on behavioral distance so that we can transfer properties even if the behavior is slightly different. We allow the system behavior to differ in both valuations of the system variables as well as timings. We give a sufficient criterion under which a system A is simulated by another system B using our notion of simulation and prove that for all formulas that are satisfied by B there is a "similar" formula that is satisfied by A.

Structure of this Paper. In Sect. 2 we give the formal basis for specifying hybrid systems and properties of those. Section 3 provides a motivating example for the study of system similarity which is examined further in Sect. 4. In Sect. 5 we give sufficient conditions under which we can assert that systems are similar and study what properties are preserved by this relation in Sect. 6.

2 Basics

To specify systems, we use a standard notion called hybrid automata. The syntax is similar to the syntax for hybrid automata originally defined by Henzinger in [13].

Definition 1. *A hybrid automaton is a tuple* $H = (U, X, L, E, F, Inv, Init)$ *where*

- *$V := U \mathbin{\dot{\cup}} X$ is a set of real-valued variables where U is the set of external variables and X contains the internal ones.*
- *L is the set of locations.*
  - *Invariants are provided by a mapping Inv of locations to predicates over variables in V.*
  - *Flows are given by F, which is a mapping of locations to predicates of the form $\bigwedge_{x \in \dot{X}} \dot{x} = e_x$ where e_x are expressions over V. Here, $\dot{X}$ denotes the derivatives of the variables in X.*
- *$E \subseteq L \times G \times L$ are discrete transitions, where G denotes predicates over $V \cup X'$. The variables in X' are valuated with the values of the variables in the post-state.*
- *$Init$ is a mapping of locations to predicates over V, which characterizes the initial condition to start in the specific location.*

We use a hybrid time model that is the cross product of the non-negative real numbers (denoted by $\mathbb{R}$ for real numbers and $\mathbb{R}^+$ for the non-negative subset) and the natural numbers (denoted by $\mathbb{N}$). The natural number component allows arbitrarily many discrete jumps at a single real-valued point in time. Further, we use $|\cdot|$ to denote the cardinality of sets or the absolute value of numbers and $\dot{\cup}$ to denote the disjoint union of sets.

The semantics of a hybrid automaton is given by a set of runs. A *run* maps a state to each point in time, which is pair of a non-negative real number and a natural number. A *state* consists of a discrete location and a valuation of the system variables. Every run of the system starts in some location and has a valuation of the variables that satisfies the initial condition. If there is a discrete jump, meaning an increase of the second argument leads to a changed valuation, then there was an edge leading from the previous to the current location, whose guard was satisfied. This also means that the current valuation of variables are subject to certain restrictions based on this guard. Continuous flows range over real-valued time-points. They change the variables based on the solution of the flow predicate. While time is passing, the variable values change continuously, and each of the valuations has to satisfy the location invariant.

Let π denote the projection to specific components of our hybrid automaton. For example $\pi_V(s)$ gives us the valuation of the variables in state s whereas $\pi_L(s)$ returns the location.

Definition 2. *The semantics of a hybrid automaton is given by a set of runs generated by a function $\Xi : ((\mathbb{R}^+ \times \mathbb{N}) \to \mathbb{R}^{|U|}) \to 2^{(\mathbb{R}^+ \times \mathbb{N}) \to L \times \mathbb{R}^{|X|}}$. Here, for some input stream ι, which is a total function $\iota : (\mathbb{R}^+ \times \mathbb{N}) \to \mathbb{R}^{|U|}$, we say the output stream ω is possible for ι, i.e. $\omega \in \Xi(\iota)$, iff there is a run, i.e. total function, $\xi(x, y) := (\iota(x, y), \omega(x, y))$ such that:*

1. *$\xi(0, 0) \models Init$*
2. *If $\xi(t, n + 1) \neq \xi(t, n)$ then there is an edge from $l = \pi_L(\xi(t, n))$ to $l' = \pi_L(\xi(t, n+1))$ and $\pi_V(\xi(t, n))$ satisfies the invariant of l and $\pi_V(\xi(t, n+1))$ satisfies the invariant of l'. Additionally, the guard on this edge is satisfied by using the values of $\xi(t, n)$ for V and $\pi_X(\xi(t, n + 1))$ for X'.*

3. *There is an upper bound on the number of discrete changes at each real-valued point in time:* $\forall t : \exists n : \forall n' > n : \xi(t, n) = \xi(t, n')$. *For each* t*, we call the smallest* n *for which this condition holds* n_t.
4. *During continuous evolutions from some real-valued time point* t *to some* t' *the location stays constant and no discrete computations change any values. They start after the discrete computations stabilized, i.e. at* (t, n_t)*, and all intermediate states* t'' *satisfy the invariant of the current location:* $\forall t : \exists t' > t : \forall t < t'' \leq t' : \pi_L(\xi(t'', 0)) = l \wedge (t'' < t' \rightarrow \forall n'' : \xi(t'', 0) = \xi(t'', n'')) \wedge \xi(t'', 0) \models Inv(l) \wedge \pi_V(\xi(t'', 0)) = s(t'' - t)$ *where* $l = \pi_L(\xi(t, n_t))$ *and* s *is a solution to the initial value problem* $s(0) = \pi_V(\xi(t, n_t))$ *of the flow predicate* $F(\pi_L(\xi(t, n)))$.

Note that this definition of the semantics explicitly excludes runs where there is an infinite number of different states at a single real-valued point in time. It, thus, ensures that every instantaneous discrete calculation comes to a result after a finite number of steps. However, Zeno behavior can still occur, as the delays between calculation steps might go to zero in the limit. As such systems would not be implementable, we assume in the following that those effects do not occur. For example the models could be altered in a way that between every two discrete transitions there is some small constant delay.

To describe properties of hybrid systems we need a logic that is able to express temporal relations of real-valued states. Therefore, we study the following real-valued real-time linear time temporal logic that we call $\mathcal{L}\natural$ in this paper.

First, let us define what a Lipschitz continuous function is.

Definition 3 (Lipschitz continuity). *We say that a function* $f : \mathbb{R}^n \rightarrow \mathbb{R}$ *is Lipschitz continuous, iff there is some constant* M *such that for all* $x_1, \ldots, x_n$*, and all* $y_1, \ldots, y_n$ *holds:*

$$|f(x_1, \ldots, x_n) - f(y_1, \ldots, y_n)| \leq M \cdot ||(x_1, \ldots, x_n), (y_1, \ldots, y_n)|| \ ,$$

where $||\cdot, \cdot||$ *denotes the Euclidean distance. We call the smallest* M *that has this property the Lipschitz constant of* f.

Now, based on this definition to restrict the possible basic terms, the syntax of our logic is defined as follows:

Definition 4 (Syntax). *The basic formulas are defined by*

$$\phi ::= x \in \mathcal{I} \mid f(x_1, \ldots, x_n) \leq 0 \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \, \mathbb{U}_{\mathcal{J}} \, \phi_2$$

where $\mathcal{I} \subseteq \mathbb{R}$, $\mathcal{J} \subseteq \mathbb{R}^+$, f *is a Lipschitz continuous function and the* x_i *are variables.*

We interpret this metric variant of LTL on runs of hybrid systems. However, as transient states of the automaton are not observable in the real world, we project the runs on a continuous domain. This gives us a valuation function which maps a value to each variable at each real-valued point in time.

Definition 5 (Valuation). *We define the valuation of a variable x at time t on a run ξ as*

$$\zeta_\xi(t,x) := \lim_{n\to\infty} \xi(t,n)|_x \ ,$$

where $y|_x$ denotes the projection of the vector y to its component associated with the variable name x.

If the run is obvious from the context we just write $\zeta(t,x)$.

Note that this definition is well defined as property 3 of our semantics of hybrid automaton demands that the number of discrete changes to the variables is finite at each real-valued point in time. For the original semantics of hybrid automaton by Henzinger [13] the limit might not exist, as a countably infinite number of transitions might be taken at a real-valued time-point and each might assign new values to the variables. However properties of these runs cannot be observed in the real world, and we, thus, drop them from our focus.

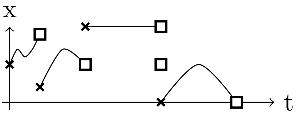


Fig. 1. Example trajectory

In Fig. 1 an example trajectory of a hybrid system is shown. Here, squares mark values that are omitted by our valuation function.

We now turn our focus to the semantics of the logical formulas. The semantics of the basic terms imposes bounds on the valuations of the system variables. This can either be done by restricting the values of a variable to be within some real-valued set or by imposing a Lipschitz continuous constraint on multiple variables. The variables are connected to the system state using the valuation function just defined. Boolean connectives are defined as usual, and the until operator provides us with the possibility to express both the temporal order of states as well as postulate time bounds on these orders. The set annotation forces the postcondition to hold at some point in time that lies within this set.

Definition 6 (Semantics). *We define for a run ξ and some $t \in \mathbb{R}^+$ the semantics of a formula ϕ by:*

$$\xi, t \models x \in \mathcal{I} \textit{ iff } \zeta(t,x) \in \mathcal{I} \tag{1}$$

$$\xi, t \models f(x_1,\ldots,x_n) \le 0 \textit{ iff } f(\zeta(t,x_1),\ldots,\zeta(t,x_n)) \le 0 \tag{2}$$

$$\xi, t \models \neg\phi \textit{ iff not } \xi, t \models \phi \tag{3}$$

$$\xi, t \models \phi \wedge \psi \textit{ iff } \xi, t \models \phi \textit{ and } \xi, t \models \psi \tag{4}$$

$$\xi, t \models \phi \, \mathbb{U}_{\mathcal{J}} \, \psi \textit{ iff } \exists t' \in \mathcal{J} : \xi, t' + t \models \psi \textit{ and } \forall t \le t'' < t' + t : \xi, t'' \models \phi \tag{5}$$

Additionally we define for a set of runs Ξ:

$$\Xi, t \models \phi \textit{ iff for all runs } \xi \in \Xi \textit{ holds } \xi, t \models \phi \tag{6}$$

A hybrid system Sys satisfies a formula denoted by $Sys \models \phi$ iff $\Xi_{Sys}, 0 \models \phi$.

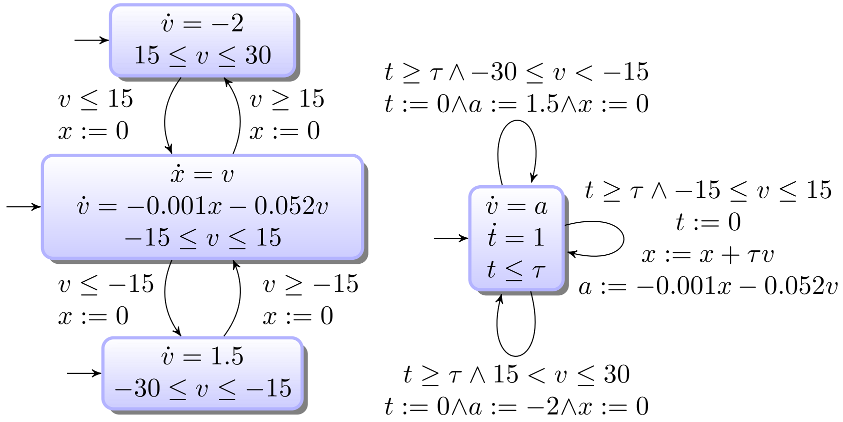
Obviously, *true* and *false* can be expressed in various ways using this logic. Additionally, we define the *eventually* modality by $\Diamond_{\mathcal{J}}\phi \equiv \mathit{true} \, \mathbb{U}_{\mathcal{J}} \, \phi$ and the *always* modality as abbreviation $\Box_{\mathcal{J}}\phi \equiv \neg\Diamond_{\mathcal{J}}\neg\phi$.

3 Running Example

To further motivate the study of similarity and to illustrate how our results can be applied, we present a specification of a cruise controller taken from [4] and provide a very basic implementation that we use for comparison.

The goal of the cruise controller is to stabilize the system at a velocity difference of $v = 0$ to some target velocity within a certain time bound if it was started with a velocity difference v between -30 and 30. If the velocity difference is below -15 the controller just chooses the maximum acceleration 1.5. In the range of -15 to 15 a proportional-integral (PI) controller takes over. It controls the acceleration proportional to the current and the accumulated velocity difference since this mode was entered. The velocity difference is accumulated by integration over v. We, here, write the integral implicitly as differential equation $\dot{x} = v$. The integral part is used by the PI controller to smoothen the velocity trajectory as it is approaching its target, i.e. a velocity difference of $v = 0$.

If the difference is above 15 the controller enforces braking with maximal deceleration of -2. Figure 2a shows the corresponding automaton. The model consists of three modes. Depending on the initial velocity difference, exactly one of these is enabled. The variable x is used to track the integral over v such that the PI control used in the central mode can access this data.



2a: Original specification

2b: Implementation

Fig. 2. Cruise controller variants

A sampling implementation is provided by the automaton depicted in Fig. 2b. Here, a single mode is sufficient and the acceleration is updated depending on the current velocity difference with a sampling rate of $\tau := 10$ time units. The two systems do not produce identical trajectories and the implementation is not a classical refinement of the specification. Still, they are similar in their behavior.

4 Similarity

To introduce a notion of similarity, we compare the observable values of the system trajectories. Like in the definition of the logic, we restrict our focus to a single valuation of variable at each real-valued point in time. Now, the idea is to say that two runs are similar if both evolve in a similar fashion by comparing variable valuations of one system with valuations of the other that might be shifted in time. This means given a valuation of the variables of one system at some point in time, there is within close distance in time a point where the valuation of the variables of the other system is close. We restrict the temporal distance by a constant ε and the spatial distance by another constant δ.

Definition 7. *For two streams $\sigma_i : \mathbb{R}^+ \times \mathbb{N} \to \mathbb{R}^p$ with $i \in \{1, 2\}$, given two non-negative real numbers ε, δ, we say that σ_1 is ε-δ-simulated by stream σ_2 (denoted by $\sigma_1 \trianglelefteq^{\varepsilon,\delta} \sigma_2$) iff there is some left-total, surjective relation $\mathfrak{r} \subseteq \mathbb{R}^+ \times \mathbb{R}^+$ with*

$$\forall (t, \tilde{t}) \in \mathfrak{r} : |t - \tilde{t}| < \varepsilon \wedge \forall (t', \tilde{t}') \in \mathfrak{r} : (t \leq t' \to \tilde{t} \leq \tilde{t}') \tag{7}$$

and

$$\forall (t, \tilde{t}) \in \mathfrak{r} : ||c(\sigma_1)(t), c(\sigma_2)(\tilde{t})|| < \delta \tag{8}$$

where for $k \in \{1, 2\}$: $c(\sigma_k)$ is defined by $c(\sigma_k)(t) := \lim_{q\to\infty} \sigma_k(t, q)$.

As $\mathfrak{r}$ allows stretching or compressing the time line, we call it a *retiming* relation. An example for such a relation is depicted in Fig. 3. The relation is motivated by the fact that slight variations in the switching points of the system as those variations should not endanger the safety of any robust real-world system.

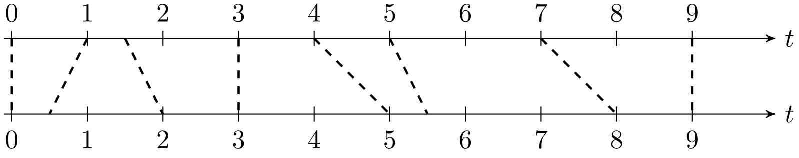


Fig. 3. Example for a retiming relation $\mathfrak{r}$

In some cases it is sufficient to use a slightly weaker notion of similarity. This can be obtained by dropping the bound on the temporal distance. If we do not impose an upper bound on the temporal distance, we can still get useful insights. When comparing systems where the timing behavior is of limited interest, to prove that, for instance, the one system works within certain spatial bounds compared to the other one, it is sufficient to use the following notion of similarity.

Definition 8. *For two streams $\sigma_i : \mathbb{R}^+ \times \mathbb{N} \to \mathbb{R}^p$ with $i \in \{1, 2\}$, given a non-negative real number δ, we say that σ_1 is weakly δ-simulated by stream σ_2 (denoted by $\sigma_1 \trianglelefteq^{\delta} \sigma_2$) iff there is some left-total, surjective relation $\mathfrak{r} \subseteq \mathbb{R}^+ \times \mathbb{R}^+$ with*

$$\forall (t, \tilde{t}) \in \mathfrak{r} : \forall (t', \tilde{t}') \in \mathfrak{r} : (t \leq t' \to \tilde{t} \leq \tilde{t}') \tag{9}$$

and (8) holds.

We apply the same definitions to output streams as well, by ignoring the location component of those. Now we introduce the main similarity notion on hybrid systems used in this paper using the notion of the stream similarity.

Definition 9. *A hybrid system A is ε-δ-simulated by another system B (denoted by $A \trianglelefteq^{\varepsilon,\delta} B$) iff for all input streams ι_A and for all input streams ι_B for which $\iota_A \trianglelefteq^{\varepsilon,\delta} \iota_B$ and for all output streams $\omega_A \in \Xi(\iota_A)$ of A, there is an output stream $\omega_B \in \Xi(\iota_B)$ of B such that $\omega_A \trianglelefteq^{\varepsilon,\delta} \omega_B$ holds. And similarly, A is weakly δ-simulated by B (denoted by $A \trianglelefteq^{\delta} B$) iff the above conditions hold for weak δ-simulations on the streams.*

Note that if a system has no input variables, then it still has a possible input stream of type $\mathbb{R}^+ \times \mathbb{N} \rightarrow \emptyset$. Therefore, in the absence of input variables, the whole relation is determined by the system outputs.

5 Determining Similarity

In this section we present a sufficient criterion for determining whether two systems are similar. We assume that the system inputs can be described by differential equations. This way we can add these differential equations to each mode and further assume inputless systems.

Now, we give an encoding of the question whether a system A is ε-δ-simulated by a system B into a two player game. The idea is that if there is a winning strategy for the second player then the systems are in simulation relation.

First, we give the general definition of hybrid games.

Definition 10 (Hybrid Game). *A* hybrid game *$HG = (S, E_c, U_c, l)$ consists of a hybrid automaton $S = (U, X, L, E, F, Inv, Init)$, a set of controllable transitions $E_c \subseteq E$, a set of controllable variables $U_c \subseteq U$, and a location $l \in L$.*

The game is played on the states of the hybrid system denoted by S. The possible moves of the first player are determined by the uncontrollable transitions $E \setminus E_c$ and the corresponding invariants and guards. The second player plays on the controllable transitions E_c. In addition, the second player always proposes a function that gives the future valuations of the variables in U_c until the next move is determined. At every state of the game each player chooses an action that is either a finite number of discrete transitions (uncontrollable transitions for the first player and controllable ones for the second player) or a time period they want to let pass. If both players choose discrete transitions then the first player gets precedence and all transitions of the first player are executed. Afterwards the transitions proposed by the second player are executed, if they are still enabled. If both players choose to let time pass, the smaller amount of time is taken. In case one chooses a discrete transition and the other one chooses to let time pass, the discrete transition gets precedence.

The first player wins, if he can force the game to enter the location l or if the second player does not have any more moves. The second player wins, if he can assert that the location l is avoided.

The case where the second player has no more moves can happen if for example the system is on the edge of an invariant region and thus a discrete transition has to happen, but no controllable transition is enabled.

To translate the question whether two systems are in simulation relation into such a hybrid game, we encode the restrictions of the simulation relation into a hybrid automaton that is able to check whether either the distance between the system states is too large, or whether we are not able to find a suitable retiming at a certain point. The retiming is modeled by speeding up/slowing down the system dynamics by multiplying them with either s for the dynamics of the first system or $2 - s$ for those of second one. As s can be altered arbitrarily we can emulate all possible retiming relations. We keep track of the temporal distance of the systems in the variable r that represents the integral over $2s - 2$. That r indeed models the temporal distance can be seen if one considers the evolution of local clocks. A clock in the first system evolves with rate s while a clock in a second system evolves with $2 - s$. In case $s = 1$ both evolve with speed 1. Else we have a clock drift of $s - (2 - s) = 2s - 2$.

The spatial distance can be checked directly. We add invariants that force the automaton to go to the *bad* location if the distance is too large. Most of the controllable transitions are only enabled as long as the system variables and timings are close. Only in cases, where the second player reacts on some action performed by an uncontrollable transition this is not directly enforced. Therefore, all uncontrollable transitions lead to a location (second component is in $\hat{L}$) where the second player might react. However, in these locations no time must pass which is enforced using the fresh clock c.

Formally this gives:

Definition 11 (Simulation Game). *Given two real numbers ε, δ, a hybrid system $A = (U_A, X_A, L_A, E_A, F_A, Inv_A, Init_A)$, and another hybrid system $B = (U_B, X_B, L_B, E_B, F_B, Inv_B, Init_B)$, we define, w.l.o.g. assuming $V_A \cap V_B = \emptyset$, a hybrid game $SG = (A \lessdot B, E_c, \{s\}, bad)$ in the following way:*

- *$A \lessdot B = (U_\lessdot, X_\lessdot, L_\lessdot, E_\lessdot, F_\lessdot, Inv_\lessdot, Init_\lessdot)$*
- *The variables of the resulting system are given by $U_\lessdot = U_A \cup U_B \cup \{s\}$ and $X_\lessdot = X_A \cup X_B \cup \{r, c\}$ where w.l.o.g. $(V_A \cup V_B) \cap \{s, r, c\} = \emptyset$ holds.*
- *The locations are given by $L_\lessdot = L_A \times (L_B \cup \hat{L}_B) \mathbin{\dot\cup} \{bad\}$, where $\hat{L}_B$ are duplicates of the original locations in L_B.*
- *Let χ be the following formula: $||x_A, x_B|| < \delta \wedge |r| < \varepsilon$, where x_A and x_B are the state vectors of the systems A and B respectively.*
- *The discrete transitions $E_\lessdot$ are the smallest set such that:*
  - *If $(l_A, \phi, l'_A) \in E_A$ then for all $l_B \in L_B$,*
  $$((l_A, l_B), \phi \wedge \chi \wedge c' = 0, (l'_A, \hat{l}_B)) \in E_\lessdot \ .$$
  - *If $(l_B, \phi, l'_B) \in E_B$ then for all $l_A \in L_A$,*
  $$((l_A, l_B), \phi \wedge \chi, (l_A, l'_B)) \in (E_\lessdot \cap E_c)$$
  and
  $$((l_A, \hat{l}_B)), \phi, (l_A, l'_B)) \in (E_\lessdot \cap E_c) \ .$$

  - *For all $l_A \in L_A$ and all $l_B \in L_B$, $((l_A, l_B), \neg\chi, bad) \in E_{\lessdot}$.*
  - *For all $l_A \in L_A$ and all $l_B \in L_B$, $((l_A, \hat{l}_B), true, (l_A, l_B)) \in (E_{\lessdot} \cap E_c)$.*

- *For all $l = (l_A, l_B) \in L_{\lessdot}$ or $l = (l_A, \hat{l}_B) \in L_{\lessdot}$ we construct $F_{\lessdot}(l)$ as $\dot{r} = 2s - 2 \wedge \dot{c} = 1 \wedge mod(F_A(l_A), s) \wedge mod(F_B(l_B), 2 - s)$ where $mod(F, x)$ alters f by multiplying the right side of each differential equation occurring in F with x.*
- *The invariants of the locations are given by $Inv_{\lessdot}$ which assigns each location (l_A, l_B) or $(l_A, \hat{l}_B)$ an invariant of the form $Inv_A(l_A) \wedge Inv_B(l_B) \wedge \chi \wedge 0 \leq s \leq 2$. If $l = (l_A, \hat{l}_B)$ we further add $c \leq 0$.*
- *$Init_{\lessdot} = \{((l_A, l_B), Init_A(l_A) \wedge Init_B(l_B)) \mid l_A \in L_A \wedge l_B \in L_B\}$*

Using this game, we can determine whether two systems stand in simulation relation as defined in Def. 9.

Theorem 1. *Given two hybrid systems A and B. If there is a winning strategy for the second player in the game $(A \lessdot B, E_c, \{s\}, bad)$ then $A \trianglelefteq^{\varepsilon,\delta} B$ holds.*

Proof. Assume that there is a winning strategy but $A \trianglelefteq^{\varepsilon,\delta} B$ does not hold. From the latter, we know that there is a run of system A such that, no matter which retiming is applied, system B cannot stay close enough. Let this run be ξ. We now construct a winning strategy for the first player using ξ. The first player chooses his actions in a way that the valuations of the variables of the first system at time t coincide with $\xi(t - r)$. The second player is not able to influence the valuations of the variables at those points, as the discrete transitions that it can choose from are those of B. He is also not able to restrict the movement of the first player, as the intermediate locations only allow him to react on actions performed by the first player and as he looses if there is a time deadlock, he can also not avoid time going to infinity. This, as there was no run of B that stays close enough to ξ, eventually leads to a state where the condition $||x_A, x_B|| < \delta \wedge |r| < \varepsilon$ is violated. In this state the first player can choose to enter the location *bad*. This contradicts the assumption that there is a winning strategy for the second player and thus concludes the proof. □

Note that the reverse implication does not hold, as the game demands a tighter coupling of the system behaviors with regard to branching than our notion of ε-δ-simulation.

Corollary 1. *If we restrict the possible moves of the second player by adding differential equations describing the evolution of s and he is still able to win the game, then the systems are in simulation relation.*

This follows directly from the fact that Theorem 1 demands the existence of a winning strategy. Now, if we can find a winning strategy for a system where the control of s is further restricted, we still can assert that there is a winning strategy for the original system.

A good strategy to use for controlling s is optimal control with the goal of minimizing the value of $||x_A, x_B||$. As the Euclidean distance contains a square root we w.l.o.g. take the square of the distance as minimization target. This yields equivalent results as the square root is a monotone transformation. For $x_A = (x_{A,1}, \dots, x_{A,n})$ and $x_B = (x_{B,1}, \dots, x_{B,n})$, the square of the distance evolves as follows:

$$\begin{aligned}\frac{d(||x_A, x_B||)^2}{dt} &= \frac{d(\sqrt{((x_{A,1} - x_{B,1})^2 + \cdots + (x_{A,n} - x_{B,n})^2)}^2)}{dt} \\ &= \frac{d((x_{A,1} - x_{B,1})^2 + \cdots + (x_{A,n} - x_{B,n})^2)}{dt} \\ &= \Sigma_{i=1}^{n}(2(x_{A,i} - x_{B,i}) \cdot (s\frac{dx_{A,i}}{dt} - (2 - s)\frac{dx_{B,i}}{dt}))\end{aligned}$$

Let s_{min} be the s that minimizes this term. Now choose s in the following way: If $r < \varepsilon \wedge s_{min} > 1$ or $r > -\varepsilon \wedge s_{min} < 1$ choose $s = s_{min}$. Otherwise choose $s = 1$. The resulting strategy, for controlling s can then be encoded into a hybrid automaton and included into the original automaton.

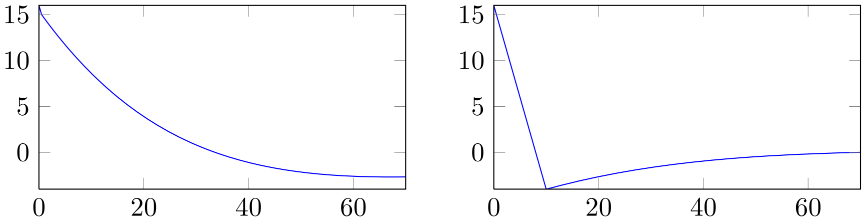
The choice of the strategy is motivated by the fact that the location *bad* can only be entered if the distances between the two systems become too large. As we might be able to trade spatial distance against temporal distance we choose to minimize the spatial distance. However, this strategy is only an heuristic as there are systems, where it is necessary to let the spatial distance increase a bit to for example unify switching timings, as the guard effects might otherwise lead to a violation of the bounds on the spatial distance.

Definition 12. *A hybrid automaton is considered* deterministic *if (1) all of its transitions are urgent, i.e. all the guards of the transitions are overlapping in a singular point with the border of its sources invariant and all trajectories of the mode are pointing outwards of the invariant region at that point, and (2) for each point in time, at most one transition is enabled.*

Let A be a hybrid automaton and B be a deterministic hybrid automaton. If we modify $A \lessdot B$ in a way that the assumptions of Corollary 1, assume that the system values are identical at the initial locations, and are able to show that on all runs of this modified version of $A \lessdot B$ the location *bad* is avoided, then, by Corollary 1, $A \trianglelefteq^{\varepsilon,\delta} B$ holds. The assumptions of Corollary 1 could, e.g., be satisfied by using the optimal control strategy. Thus, we can use a model checker (e.g. FOMC [3], PHAVer [8], SpaceEx [9], or HSolver [20], depending on the system class and complexity) to search for a certificate for the fact that *bad* is unreachable. Provided that suitable inductive invariants can be found, we could also prove that the trajectories of these two uncontrolled systems stay close using a theorem prover for hybrid systems like KeYmaera [19].

On a similar line of thought, if one can prove that the systems stabilize within a certain time, as it is the case for our running example, it also possible to use bounded model checking up until the time of stabilization. This could be performed, e.g., using iSAT [10].

Let us now consider our example presented in Sect. 3 with respect to these results. Using MATLAB *Simulink* we can determine that the maximum distance



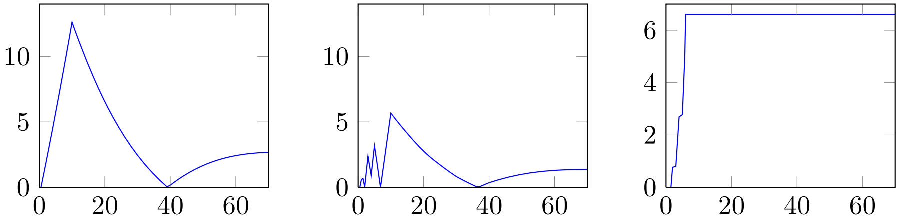
4a: Velocity under continuous control

4b: Velocity under sampled control

Fig. 4. Simulated velocities over time

between these two systems if started in the initial state $v = 16$ is 12.7. The velocity trajectories of the two systems are shown in Fig. 4a and Fig. 4b. A plot of the resulting differences is depicted in Fig. 5a.

Using optimal control to keep the distance between the two systems minimal as long as the retiming bounds allow and the fact that both systems stabilize at $v = 0$ we can also show that there is a 5-6.61-simulation, if the initial region is restricted to this singular point $v = 16$. This enables us to transfer more knowledge from one model to the other than the previous observation. The resulting trajectory corresponds to applying a retiming relation that results in the temporal differences depicted in Fig. 5c to the original trajectories and we get the smaller differences (compared to Fig. 5a) shown in Fig. 5b.



5a: Without retiming

5b: With retiming

5c: Temporal differences

Fig. 5. Maximal differences of the system velocities over time

6 Preservation of Logical Properties

Now that we have identified sufficient conditions for our simulation relation, we study what properties are preserved by the relation.

The similarity notion defined in Def. 9 can be used to transfer properties from one system to another. As we have upper bounds on the deviations we can use those to weaken the formulas to be sure that they still hold in similar systems.

Theorem 2. *If hybrid systems A and B satisfy $A \trianglelefteq^{\varepsilon,\delta} B$ and $B \models \phi$ then $A \models \phi^{+\delta}_{+\varepsilon}$ where $\phi^{+\delta}_{+\varepsilon} := re_{\varepsilon,\delta}(\phi)$ and $re_{\varepsilon,\delta}$ is defined by:*

- *$re_{\varepsilon,\delta}(x \in \mathcal{I}) := x \in \mathcal{I}'$, where $\mathcal{I}' = \{a \mid \exists b \in \mathcal{I} : a \in [b-\delta, b+\delta]\}$.*
- *$re_{\varepsilon,\delta}(f(x_1,\ldots,x_n) \leq 0) := f(x_1,\ldots,x_n) - \delta \cdot M \leq 0$ where M is the Lipschitz constant for f.*
- *$re_{\varepsilon,\delta}(\neg\phi) := \neg ro_{\varepsilon,\delta}(\phi)$.*
- *$re_{\varepsilon,\delta}(\phi \wedge \psi) := re_{\varepsilon,\delta}(\phi) \wedge re_{\varepsilon,\delta}(\psi)$.*
- *$re_{\varepsilon,\delta}(\phi \, \mathbb{U}_{\mathcal{J}} \, \psi) := re_{\varepsilon,\delta}(\phi) \, \mathbb{U}_{\mathcal{J}'} \, re_{\varepsilon,\delta}(\psi)$, where $\mathcal{J}' = \{a \mid \exists b \in \mathcal{J} : a \in [b - \varepsilon, b+\varepsilon] \cap [0,\infty)\}$.*

The transformation function $ro_{\varepsilon,\delta}$ is given by:

- *$ro_{\varepsilon,\delta}(x \in \mathcal{I}) := x \in \mathcal{I}'$, where $\mathcal{I}' = \{a \mid \exists b \in \mathcal{I} : a \in [b+\delta, b-\delta]\}$.*
- *$ro_{\varepsilon,\delta}(f(x_1,\ldots,x_n) \leq 0) := f(x_1,\ldots,x_n) + \delta \cdot M \leq 0$ where M is the Lipschitz constant for f.*
- *$ro_{\varepsilon,\delta}(\neg\phi) := \neg re_{\varepsilon,\delta}(\phi)$.*
- *$ro_{\varepsilon,\delta}(\phi \wedge \psi) := ro_{\varepsilon,\delta}(\phi) \wedge ro_{\varepsilon,\delta}(\psi)$.*
- *$ro_{\varepsilon,\delta}(\phi \, \mathbb{U}_{\mathcal{J}} \, \psi) := ro_{\varepsilon,\delta}(\phi) \, \mathbb{U}_{\mathcal{J}'} \, ro_{\varepsilon,\delta}(\psi)$, where $\mathcal{J}' = \{a \mid \exists b \in \mathcal{J} : a \in [b + \varepsilon, b-\varepsilon] \cap [0,\infty)\}$.*

As before $\mathcal{I} \subseteq \mathbb{R}$ and $\mathcal{J} \subseteq \mathbb{R}^+$ hold.

The function $re_{\varepsilon,\delta}$ is applied if the current subformula is in a context of an even number of negation, whereas the function $ro_{\varepsilon,\delta}$ is applied to subformulas under an odd number of negations. Note that if the set indexing an until operator becomes empty, the formula is trivially false.

Our notion of similarity can be seen as a decrease of the resolution of the image we have of the system behavior thus blurring the borders. If we originally knew that at some time between t and t' some event would happen, we now have to account for the timing deviations that might occur. Thus, if the event originally happened at time t it might now occur in the worst case already at $t - \varepsilon$. If it originally occurred at time t', the worst case we have to consider is that it now might occur as late as $t' + \varepsilon$. This widens the set of possible time points for the event, thus reducing our knowledge about exact timings. A similar effect happens on the variable valuations.

This theorem can be proven by induction over the formula structure and the time. The induction base is formed by showing that the modified formulas hold at time 0. One important argument for this is the fact that $(0,0) \in \mathfrak{r}$ and of course the bounds on the deviations. Now the only operation that modifies the time at which the formulas are evaluated is the until operator. As stated in the previous paragraph, from the fact that the systems are in simulation relation, we know that the postcondition was originally satisfied during some point in the set annotation, the modified version of the formula might now hold a bit earlier or a bit later but is forced to hold at some point. Up until this point, all the values have to be similar to those originally satisfying the precondition.

The other crucial point is to prove that negated formulas can be transfered. This can be shown by another induction. Again the difficult part is to show that the proposition holds for the until operator. If we assume, the modified formula would not hold, we can use our relation $\mathfrak{r}$ that gives us a point in time where the original pre- and postconditions touches. This, however, is a contradiction to the fact that the until is not satisfied for the original system.

Using the weaker version of similarity we can also transfer some properties. The version is weaker with respect to knowledge about timing. Those timings are only present in the intervals indexing the until-operators in the formulas. Thus weakening these operators by replacing those intervals by $[0, +\infty)$ removes all exact timing informations. Only temporal properties are preserved in that case, e.g. we could still retain knowledge about event orders.

Theorem 3. *If for hybrid systems A and B holds $A \trianglelefteq^{\delta} B$ and $B \models \phi$, where ϕ does not contain any until-operations in a negative context, then $A \models \phi^{+\delta}_{\sim}$ where $\phi^{+\delta}_{\sim} = w(re_{0,\delta}(\phi))$ and w replaces the index of every until operator by $\mathbb{R}^{+}$.*

The proof follows easily from the proof for Theorem 2 by altering the induction steps for the until operator. Unfortunately, we here loose to much information about the timings to keep knowledge about until operations in a negative context, i.e. under an odd number of negations, thus the restriction to positive contexts in this theorem.

7 Summary

In this paper we presented our notion of similarity for hybrid systems that we call ε-δ-simulation. We have given a translation of the question whether one system simulates another into hybrid games and given sufficient conditions under which a winning strategy for these games exists. Further, we have studied what properties are preserved under this notion of similarity.

Currently, we can only determine similarity of a restricted class of models and the approach itself has a large complexity making it easier in many cases to check the properties one wants to transfer on the implementation directly. For future work, we will study how we can incorporate system decompositions into our approach. For this goal, we need to find a way to determine whether two open-loop systems are in simulation relation.

Acknowledgments. We like to thank Ernst-Rüdiger Olderog and Anders Ravn for the fruitful discussions, and Sven Linker, Johannes Faber, and Mani Swaminathan for reading preliminary versions of this paper.

References

1. Alur, R., Henzinger, T.A.: Logics and models of real time: A survey. In: Proceedings of the Real-Time: Theory in Practice, REX Workshop, London, UK, pp. 74–106. Springer, Heidelberg (1992)
2. Bouyer, P., Brihaye, T., Chevalier, F.: O-minimal hybrid reachability games. Logical Methods in Computer Science 6(1) (2009)
3. Damm, W., Dierks, H., Disch, S., Hagemann, W., Pigorsch, F., Scholl, C., Waldmann, U., Wirtz, B.: Exact and fully symbolic verification of linear hybrid automata with large discrete state spaces. Science of Computer Programming, Special Issue on Automated Verification of Critical Systems (to appear, 2011)

4. Damm, W., Dierks, H., Oehlerking, J., Pnueli, A.: Towards component based design of hybrid systems: Safety and stability. In: Manna, Z., Peled, D. (eds.) Time for Verification. LNCS, vol. 6200, pp. 96–143. Springer, Heidelberg (2010)
5. Davoren, J.M.: Epsilon-tubes and generalized skorokhod metrics for hybrid paths spaces. In: [17], pp. 135–149
6. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010)
7. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theor. Comput. Sci. 410(42), 4262–4291 (2009)
8. Frehse, G.: Phaver: algorithmic verification of hybrid systems past hytech. STTT 10(3), 263–279 (2008)
9. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011)
10. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. Journal on Satisfiability, Boolean Modeling and Computation 1, 209–236 (2007)
11. Girard, A., Julius, A.A., Pappas, G.J.: Approximate simulation relations for hybrid systems. Discrete Event Dynamic Systems 18(2), 163–179 (2008)
12. Girard, A., Pola, G., Tabuada, P.: Approximately bisimilar symbolic models for incrementally stable switched systems. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 201–214. Springer, Heidelberg (2008)
13. Henzinger, T.A.: The theory of hybrid automata. In: LICS, pp. 278–292. IEEE CS Press, Los Alamitos (1996)
14. Henzinger, T.A., Horowitz, B., Majumdar, R.: Rectangular hybrid games. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 320–335. Springer, Heidelberg (1999)
15. Henzinger, T.A., Majumdar, R., Prabhu, V.S.: Quantifying similarities between timed systems. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 226–241. Springer, Heidelberg (2005)
16. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems 2(4), 255–299 (1990)
17. Majumdar, R., Tabuada, P. (eds.): HSCC 2009. LNCS, vol. 5469. Springer, Heidelberg (2009)
18. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)
19. Platzer, A., Quesel, J.D.: Keymaera: A hybrid theorem prover for hybrid systems (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 171–178. Springer, Heidelberg (2008)
20. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation based abstraction refinement. ACM Journal in Embedded Computing Systems 6(1) (2007)
21. Stauner, T.: Discrete-time refinement of hybrid automata. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 407–420. Springer, Heidelberg (2002)

22. Thrane, C.R., Fahrenberg, U., Larsen, K.G.: Quantitative analysis of weighted transition systems. J. Log. Algebr. Program. 79(7), 689–703 (2010)
23. Tomlin, C., Lygeros, J., Sastry, S.: A Game Theoretic Approach to Controller Design for Hybrid Systems. Proceedings of IEEE 88, 949–969 (2000)
24. Vladimerou, V., Prabhakar, P., Viswanathan, M., Dullerud, G.E.: Stormed hybrid games. In: [17], pp. 480–484
25. Wulf, M.D., Doyen, L., Raskin, J.F.: Almost asap semantics: from timed models to timed implementations. Formal Asp. Comput. 17(3), 319–341 (2005)

Exact Incremental Analysis of Timed Automata with an SMT-Solver

Bahareh Badban and Martin Lange*

School of Elect. Eng. and Computer Science, University of Kassel, Germany

Abstract. Timed automata as acceptors of languages of finite timed words form a very useful framework for the verification of safety properties of real-time systems. Many of the classical automata-theoretic decision problems are undecidable for timed automata, for instance the inclusion or the universality problem. In this paper we consider restrictions of these problems: universality for deterministic timed automata and inclusion of a nondeterministic one by a deterministic one. We then advocate the use of SMT solvers for the exact incremental analysis of timed automata via these problems. We stratify these problems by considering domains of timed words of bounded length only and show that each bounded instance is in (co-)NP. We present some experimental data obtained from a prototypical implementation measuring the practical feasibility of the approach to timed automata via SMT solvers.

1 Introduction

Timed automata as introduced in [3] are one of the most well-established models for the specification and verification of systems in which events can happen arbitrarily close in time [14].

The real numbers present an adequate model for continuous time; systems in which such timing aspects need to be modeled are therefore also called real-time systems. Execution traces of real-time systems can be modeled by timed words—sequences of events which are attached to the time at which they occur. Timed automata then act as acceptors of languages of timed words for instance, and various verification problems on real-time systems can be phrased as classic automata-theoretic problems. For instance the classic problem of determining whether an implementation satisfies a specification can be modeled as a language inclusion problem on timed automata.

Real-time aspects introduce additional complexity. The state space of a timed automaton is in fact infinitely large due to the infinitely many moments in time that the execution along a timed word presents. While emptiness for finite automata is NLOGSPACE-complete, and universality (is $L(\mathcal{A}) = \Sigma^*$?) and inclusion (is $L(\mathcal{A}) \subseteq L(\mathcal{B})$?) are PSPACE-complete [3]; emptiness for timed automata

* The European Research Council has provided financial support under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 259267.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 177–192, 2011.

is already PSPACE-complete and universality and inclusion even are undecidable [3]. This has always limited the analysis of such automata to a great extent. To circumvent this problem extensive research has been done, in which certain limitations are imposed on the structure of the automata, for instance by restricting the resources that a timed automaton has access to.

In this paper we consider the problems of language universality and inclusion for timed automata. In order to obtain effective algorithmic solutions we restrict the universality problem to deterministic timed automata, and in the inclusion problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ we require that $\mathcal{B}$ is deterministic. In Sect. 2 we recall very simple constructions that embed the former as well as the emptiness problem for nondeterministic timed automata into the latter. Because of this we can restrict our attention to this inclusion problem.

In Sect. 3 we then develop an incremental approach to these three problems using an idea that has been developed for bounded model checking: by limiting the space of possible witnesses or counterexamples to the emptiness, universality or inclusion problem one can obtain a computationally easier problem. Formally, we define bounded variants of these three problems which are obtained by adding a further input parameter k. The bounded emptiness problem then asks whether or not there is a word in the language that has length at most k. Bounded variants of the two other problems are defined accordingly. We first give a lower complexity bound on the emptiness and inclusion problem showing that they remain at least NP-hard (for unarily encoded additional input parameter k). A corresponding co-NP-lower bound for the universality problem cannot be presented for lack of space but does exist.

In Sect. 4 we first develop a generic propositional logic over integer and real-valued constraints and show that its satisfiability problem is in NP. We then present an encoding of the bounded inclusion problem into the satisfiability problem for this logic. Encodings of the two other bounded problems can easily be obtained from this one. This shows NP-completeness of the bounded emptiness and inclusion problems as well as inclusion in co-NP of the bounded universality problem; compare this to the PSPACE-completeness of the unbounded versions. Hence, the transition from the unbounded to the bounded problems reduces the complexity measurably but it also is the case that the bounded problems are not necessarily solvable in polynomial time (which would make the approach via satisfiability problems questionable).

This generic propositional logic can easily be embedded into logics that are supported by many modern SMT solvers. An SMT ("satisfiability modulo theories") is a satisfiability checker for predicate logics that are obtained by expanding propositional logic with a so-called theory, for example the theory of arithmetic over the real numbers.

SMT solvers extend SAT solvers which can only check plain propositional formulas for satisfiability but cannot handle numbers or interpreted function symbols like addition for instance. This does not mean that SAT solvers cannot be used for satisfiability problems using natural numbers for example: if their domain is bounded then they can easily be modeled by propositional variables based on the bit representation. This is not restricted to natural numbers. Even timed automata analysis which inherently relies on adding real numbers has been done with SAT solvers [12]. This requires some sort of abstraction though, for instance by approximating real numbers up to a feasible precision which enables representations with a fixed number of bits. In fact, this is nothing more than the region-based abstraction technique seen in classical algorithms on timed automata.

Here we argue that SMT solvers over theories incorporating simple real arithmetic form a promising alternative to the analysis of timed automata using SAT solvers. In effect, they offer the following advantages.

1) SMT solvers can get rid of the need for abstraction techniques. The abstraction needed to solve a problem over a continuous domain with a discrete step algorithm is removed from the timed automata analysis and entirely handled by the SMT solver. This simplifies correctness proofs on the timed automata side a lot as the proofs contained in this paper show. We emphasize the lack of need for such abstractions by calling the SMT-based analysis "exact".
2) SMT solvers are not restricted to problems of a certain complexity like SAT solvers are restricted to problems in NP (unless one accepts exponentially large encodings). In this paper we only consider a very weak fragment of logics that can be handled by modern SMT solvers: quantifier-free logic over very simple linear integer or real-valued inequalities. This is enough for the problems on timed automata considered here. However, SMT solvers offer a lot more in terms of available predicate logics. The question of which other problems on timed automata can be tackled (not necessarily "solved" because of decidability issues) using SMT solvers needs to be answered in future investigations.
3) Research in SMT solving is catching up with SAT solving. For example, incremental solving—the possibility of adding and deleting constraints after solving—is available for some SMT solvers as well. Incrementality is what makes the transition from the unbounded emptiness, universality and inclusion problem to the bounded variant a feature rather than a weakness [20].

In Sect. 5 we examine the practical feasibility of using SMT solvers for timed automata analysis by reporting on some experimental results obtained from a prototype implementation using the SMT solver Z3 [1]. We conclude the paper in Sect. 6 with ideas for future work on top of the one presented here.

Related Work. Many works also consider restricted cases in which the universality or inclusion problem becomes decidable [2, 4, 18, 19]. While we also restrict these problems (to some deterministic automata), the purpose of doing so is entirely different. We are not primarily concerned with obtaining a decidable subcase. It is just that this decidable subcase can be handled with a relatively simple translation into the SMT framework, and the complexity considerations in this paper are done in order to classify the obtained subcase accordingly. The NP-hardness lower bound shows that there is no known way of solving the problem efficiently. The NP upper bound shows that the analysis problems fit into a relatively simple fragment of quantifier-free predicate logic over the integers and the reals.

The works that are closely related to the one presented here are probably [6, 17] where the authors extend some existing SAT-solvers in order to verify timed automata against reachability properties specified as LTL formula. This is when, in this paper we compare the (generalized) timed automata with each other to investigate properties like language inclusion and universality as well. Our approach is also different from [12, 21] where the authors present some encoding of the emptiness problem for networks of timed automata into SAT. The main difference between these and the work presented here does not lie in the extension to asynchronous networks. It is not hard to see that the encoding presented here can be extended to networks of timed automata as well. The main difference is, as argued above, the use of SMT instead of SAT solving technology. The simplification due to lack of need for explicit discretisation is imminent when comparing the correctness proofs here with the ones in there.

Timed automata as a subclass of hybrid automata can of course be dealt with using the approaches introduced in [5, 13]. These methods are designed in such a way that they can handle the non-linear constraints in hybrid automata efficiently. However, our intention is to provide a technique that is specifically designed for timed automata, whose clock constraints in a generalized form only compare the difference of two variables with a constant value, e.g. $x - y \leq 2$.

2 Timed Automata

2.1 Syntax, Semantics, Runs

A timed automaton consists of a finite state automaton together with a finite set of clocks. Clocks are non-negative real valued variables which keep track of the time delay since the last reset. A finite state automaton describes the system control states and its discrete transitions. All clocks are initially set to 0, and evolve at the same speed. A configuration of the system is given by the current control location of the automaton and the value of each clock, denoted $\langle q, v \rangle$, where q is the control location and v is the valuation function which assigns to each clock its current value. Transitions are enabled by *guards* which are constraints on clock values. The language $\Phi(X)$ of all constraints over a set of clocks X is given by

$$g ::= \texttt{true} \mid x \lhd c \mid x - y \lhd \mathtt{c} \mid \neg g \mid g \wedge g$$

where $x, y \in X$, $\mathtt{c} \in \mathbb{N}$ and $\lhd \in \{\leq, <\}$. We will also use other Boolean and arithmetic operators that are definable using the ones above like $g_1 \vee g_2 := \neg(\neg g_1 \wedge \neg g_2)$, etc.

The definition above, as called in the literature [8, 10] *diagonal* constraint, renders timed automata more expressive than its classical variant [3, 9] where atomic constraints contain only one clock variable, e.g. $x \leq c$.

A valuation for a set X of clocks is a function $v : X \to \mathbb{R}_{\geq 0}$, assigning non-negative real values to the clocks. The satisfication of a clock constraint over

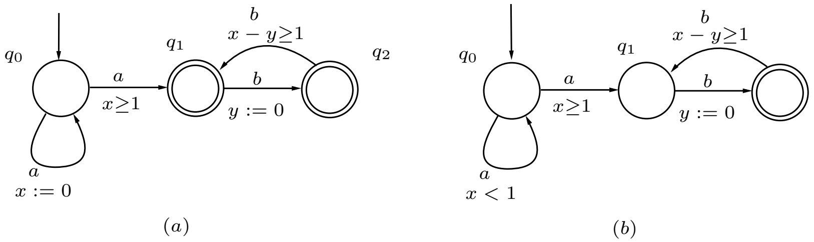


Fig. 1. a) a nondeterministic timed automaton. b) a deterministic timed automaton.

X by some valuation over X is defined straight-forwardly by induction on the structure of the constraint, e.g. $v \models x \lhd \mathsf{c}$ iff $v(x) \lhd c$ for $\lhd \in \{\leq, <\}$. A constraint g is satisfiable if there is a v s.t. $v \models g$. Otherwise it is unsatisfiable.

A (nondeterministic) *timed automaton* (TA in short) is a tuple $\langle L, L_0, L_F, \Sigma, X, E\rangle$ where L is a finite set of locations, $L_0 \subseteq L$ is a set of initial locations, $L_F \subseteq L$ is a set of final locations, Σ is a finite alphabet called *events*, X is a finite set of clock variables, and $E \subseteq L \times \Phi(X) \times \Sigma \times 2^X \times L$ is a finite set of *symbolic* transitions.

There are two kinds of concrete transitions in a timed automaton, *delay* and *discrete* ones. A delay transition lets time elapse, in which case the value of all clocks will increase accordingly. For instance, a delay transition of some non-negative time $t \in \mathbb{R}_{\geq 0}$ transforms state $\langle q, v\rangle$ into state $\langle q, v + t\rangle$, where $v + t$ assigns to each clock x the value $v(x) + t$. It is denoted $\langle q, v\rangle \xrightarrow{t} \langle q, v + t\rangle$.

A discrete transition can change the control location. There is a discrete transition with event a from state $\langle q, v\rangle$ to state $\langle q', v'\rangle$, denoted by $\langle q, v\rangle \xrightarrow{a} \langle q', v'\rangle$, if there is a symbolic transition $(q, g, a, R, q') \in E$ s.t. the clock valuation v satisfies the guard g, and v' is obtained from v by setting all clocks in R to 0.

It should be clear that two successive delay transitions with time delays t_1 and t_2 can be combined into a single delay. Furthermore, a delay transition followed by a discrete transition can be explained as a single transition based on a symbolic transition $(q, g, a, R, q') \in E$ by simply requiring that the guard g is satisfied by the clock valuation obtained from adding the delay to the current valuation: we have $\langle q, v\rangle \xrightarrow{t,a} \langle q', v'\rangle$ if $\langle q, v\rangle \xrightarrow{t} \langle q, v''\rangle$ and $\langle q, v''\rangle \xrightarrow{a} \langle q', v'\rangle$ for some (necessarily unique) valuation v''. In the following we will consider only this kind of combined delay-discrete transition to explain the operational semantics.

A *timed word* of length k over Σ is a finite sequence $(a_0, t_0), (a_1, t_0 + t_1), \ldots, (a_{k-1}, t_0 + \cdots + t_{k-1})$, where for each $0 \leq i < k :\ a_i \in \Sigma$ and $t_i \in \mathbb{R}_{\geq 0}$. The set of all timed words is denoted by $T\Sigma^*$.

Timed automata have runs on timed words just like finite automata have runs on ordinary finite words. Each event-delay pair of a timed word causes a delay-discrete transition in the automaton. As usual, runs start in initial states and are accepting if they reach a final state at the end of the input word.

Given a timed automaton $\mathcal{A} = \langle L, L_0, L_F, \Sigma, X, E\rangle$ and a timed word $w = (a_0, t_0), (a_1, t_0+t_1), \ldots (a_{k-1}, t_0+\cdots+t_{k-1})$ over Σ, a run of $\mathcal{A}$ on w is a sequence

$$\langle q_0, v_0\rangle \xrightarrow{t_0,a_0} \langle q_1, v_1\rangle \xrightarrow{t_1,a_1} \ldots \xrightarrow{t_{k-1},a_{k-1}} \langle q_k, v_k\rangle$$

where for all $0 \le i < k$ there is a $(q_i, g, a_i, R, q_{i+1}) \in E$ for some g and a such that: $v_i + t_i \models g$, and for each $x \in X$ we have $v_{i+1}(x) = 0$ if $x \in R$, and $v_{i+1}(x) = v_i(x) + t_i$ otherwise. The run is accepting if additionally we have $q_0 \in L_0$ and $q_k \in L_F$. The language $L(\mathcal{A})$ of the TA $\mathcal{A}$ is the set of all timed words for which there is an accepting run of $\mathcal{A}$.

Example 1. The timed word $w = (a, 1), (a, 2.5), (b, 2.7), (b, 2.8)$ is accepted by the TA of Fig. 1 (a) which is witnessed by the run

$$\langle q_0, 0, 0\rangle \xrightarrow{1,a} \langle q_0, 0, 1\rangle \xrightarrow{1.5,a} \langle q_1, 1.5, 2.5\rangle \xrightarrow{0.2,b} \langle q_2, 1.7, 0\rangle \xrightarrow{0.1,b} \langle q_1, 1.8, 0.1\rangle$$

where $\langle q, t, t'\rangle$ represents the state $\langle q, v\rangle$ for which $v(x) = t$ and $v(y) = t'$. The language of this automaton can be described as $L(\mathcal{A}) = \{(a, t_0), \ldots (a, t_i), (b, t'_1), \ldots, (b, t'_j) \mid t_i \ge 1 \text{ and } i, j \ge 0\}$.

A TA is *deterministic*, DTA in short, when L_0 has only one element and each two transitions with same source location and same label have disjoint guards. I.e., if $(q, g, a, R, q') \in E$ and $(q, g', a, R', q'') \in E$ then $g \wedge g'$ is unsatisfiable. A DTA $\mathcal{B}$ is *complete* if for each timed word $w \in T\Sigma^*$ there is a run of $\mathcal{B}$ over w.

It is not hard to see that every DTA $\mathcal{B} = (L, q_0, L_F, \Sigma, X, E)$ can be made complete by adding a new non-final control location $q_\perp$ and transitions $(q_\perp, \texttt{true}, a, \emptyset, q_\perp)$ for every $a \in \Sigma$ as well as $(q, \bigwedge_{g\in G_{q,a}} \neg g, a, \emptyset, q_\perp)$ for every $q \in L$, and $a \in \Sigma$ where $G_{q,a} = \{g \mid \exists R.\exists q'.(q, g, a, R, q') \in E\}$. We therefore assume that DTA are always complete.

2.2 Three Decision Problems on TA and DTA

We consider three decision problems for timed automata.

1. *Non-emptiness for TA* (nEMPTY) is: given a TA $\mathcal{A}$, is $L(\mathcal{A}) \neq \emptyset$?
2. *Universality for DTA* (dUNIV) is: given a DTA $\mathcal{B}$, is $L(\mathcal{B}) = T\Sigma^*$?
3. *TA-DTA inclusion* (ndINCL) is: given a TA $\mathcal{A}$ and a DTA $\mathcal{B}$, is $L(\mathcal{A}) \subseteq L(\mathcal{B})$?

Note that more general problems like universality or inclusion between a TA and a TA are undecidable [3]. It is not difficult to see that ndINCL subsumes the two other problems: the universal language can easily be recognised by a one-state TA $\mathcal{A}_{univ}$, and the empty language can also be recognised by a one-state TA $\mathcal{A}_{empty}$ which is in fact also a DTA. Then we have $L(\mathcal{B}) \neq \emptyset$ iff $L(\mathcal{B}) \subseteq L(\mathcal{A}_)$ for any TA $\mathcal{B}$, and we have $L(\mathcal{B}) = T\Sigma^*$ iff $L(\mathcal{A}_{univ} \subseteq L(\mathcal{B})$ for any DTA $\mathcal{B}$.

Proposition 1 ([3]). *There are linear-time reductions from* nEMPTY *and* dUNIV *to* ndINCL*.*

Consequently, we can concentrate on ndINCL for the remainder of this paper, and any method for this problem easily induces methods for nEMPTY and dUNIV as well.

3 Bounded Decision Problems

Bounded versions restrict the length of witnesses or counterexamples to a solution by an additional parameter. The *bounded TA-DTA inclusion problem* (NDINCL$^{\leq}$) for instance is the following: given a TA $\mathcal{A}$, a DTA $\mathcal{B}$ and a $k \in \mathbb{N}$, is every timed word *of length at most* k that is accepted by $\mathcal{A}$ also accepted by $\mathcal{B}$? In that case we write $L(\mathcal{A}) \subseteq_k L(\mathcal{B})$. A *witness* (for non-inclusion) is a timed word w s.t. $|w| \leq k$, $w \in L(\mathcal{A})$ and $w \notin L(\mathcal{B})$.

The following is easy to see. It shows how the bounded TA-DTA inclusion problem can be used in order to approximate the TA-DTA inclusion problem.

Proposition 2. *Let $\mathcal{A}$ be a TA, $\mathcal{B}$ be a DTA. Then $L(\mathcal{A}) \subseteq L(\mathcal{B})$ iff for all $k \in \mathbb{N}$: $L(\mathcal{A}) \subseteq_k L(\mathcal{B})$.*

The *bounded TA emptiness problem* and the *bounded DTA universality problem* are defined in the same way by adding an input parameter k and asking for words of length at most k which are (not) in the language of the given TA, resp. DTA.

It is not hard to see that the constructions in the proof of Prop. 1 go through for the bounded versions of the three considered problems there. The bounding parameter always remains the same. We therefore state the following without an extra proof.

Theorem 1. *There are linear-time reductions from* NEMPTY$^{\leq}$ *and* DUNIV$^{\leq}$ *to* NDINCL$^{\leq}$*.*

It is known that the three decision problems NEMPTY, DUNIV and NDINCL are all PSPACE-complete [3, 4].[1] We will show that bounding these problems makes them computationally easier: NEMPTY$^{\leq}$ and NDINCL$^{\leq}$ are NP-complete. For the upper bound we consider a more general satisfiability problem in the next section which will also be used to obtain implementations. It can also be used to show that DUNIV$^{\leq}$ is in co-NP. Here we prove the NP-lower bounds. DUNIV$^{\leq}$ is also co-NP-hard which can be shown by a reduction from the complement of NEMPTY$^{\leq}$.

Theorem 2. NEMPTY$^{\leq}$ *and* NDINCL$^{\leq}$ *are NP-hard for a singleton alphabet and two clocks already.*

Proof. We prove the claim for NEMPTY$^{\leq}$ by a reduction from the well-known Hamiltonian path problem. Given a directed graph $G = (V, E)$ and a node $u \in V$, is there a path starting in u that visits each vertex exactly once? This problem is known to be NP-complete [15]. The result then follows for NDINCL$^{\leq}$ immediately with Thm. 1.

Let $G = (V, E)$ and $u_0 \in V$ be given. W.l.o.g. we assume $V = \{0, \ldots, |V|-1\}$. We build a TA $\mathcal{A}_G$ that has (almost) the same transition structure as G and

[1] [3] states PSPACE-completeness of DUNIV but only shows the upper bound by a reduction to NDINCL. DUNIV is also PSPACE-hard though: it is possible (but more complicated) to reduce NDINCL to DUNIV in polynomial time.

uses two clocks: x is used to enforce a certain time delay, namely exactly time 2^i in state i; and y is used to measure the overall time used in a run. We only add a single final state which is reachable whenever time $2^n - 1$ has passed.

Thus, let $\mathcal{A}_G = (V \cup \{\mathsf{fin}\}, \{u_0\}, \{\mathsf{fin}\}, \{a\}, \{x, y\}, E')$ where

$$E' \ := \ \{(i, (x = 2^i), a, \{x\}, j) \mid (i, j) \in E\} \cup \{(i, (y = 2^{|V|} - 1), a, \emptyset, \mathsf{fin}) \mid i \in V\}$$

Finally, let $k_G := |V| + 1$. It should be clear that this construction is polynomial. Note that the representation of 2^i for instance only requires i bits. We now claim that G has a Hamiltonian path starting in u iff $L(\mathcal{A}_G)$ contains a word of length at most k_G.

"$\Rightarrow$" Let $n = |V|$ and suppose that $u_0, \ldots, u_{n-1}$ is a Hamiltonian path. It is not hard to see that $\mathcal{A}_G$ has an accepting run on the timed word $(u_0, 2^{u_0}), \ldots, (u_{n-1}, 2^{u_{n-1}}), (\mathsf{fin}, 0)$. The delay 2^{u_i} in the i-th part of this word is exactly what is needed in order to enable the guard on a transition leaving location u_i. Furthermore, since every state is visited exactly once, we have $\sum_{i=0}^{n-1} 2^{u_i} = 2^n - 1$ which enables the transition to the final state in the end.

"$\Leftarrow$". Suppose $(u_0, t_0), \ldots, (u_{n-1}, t_{n-1}), (\mathsf{fin}, t_n)$ is an accepting run of some word in $L(\mathcal{A}_G)$ and $n \leq k_G$. Since fin is only reachable by enabling the guard $y = 2^{|V|} - 1$, and no transition ever resets the clock y, we must have $\sum_{i=0}^{n-1} t_i = 2^{|V|} - 1$. Furthermore, every transition resets the clock x, and every transition out of some state i is only possible after a delay of exactly 2^i in this state. Hence, all the t_i for $i = 0, \ldots, n-1$ are powers of 2, add up to $2^{|V|} - 1$, and there are at most $|V|$ of them. This is only possible if $n - 1 = |V|$ and each of the 2^i occurs exactly once in this sum. But this is only possible if the accepting run visits each location exactly once, i.e. forms a Hamiltonian path in the underlying graph G. □

4 Incremental Encodings of Some Decision Problems

4.1 A Generic Constraint Logic

We consider a generic propositional logic which can have constraints over integer- and real-valued variables as literals, and present an encoding of the bounded inclusion problems into this logic. Decidability and implementability of the bounded inclusion problem can then be tackled by translating this logic into some known and specialised formalism.

Let $\mathcal{V}^{\mathbb{Z}}$ and $\mathcal{V}^{\mathbb{R}}$ be two disjoint sets of natural and real-valued variables. We use two different fonts in order to distinguish variables and constant values: $x, y, \ldots$ are variables, $\mathsf{b}, \mathsf{c}, \ldots$ are constants. We also assume that it will always be clear from the context what the type of a given variable is.

An *integer constraint* is of the form $\mathsf{b} \cdot x \leq \mathsf{c}$ where $x \in \mathcal{V}^{\mathbb{Z}}$, $\mathsf{b} \in \{1, -1\}$, and $\mathsf{c} \in \mathbb{Z}$. A *real constraint* is of the form $\sum_{i=1}^{m} \mathsf{b}_i \cdot x_i \lhd \mathsf{c}$ where $\lhd \in \{<, \leq\}$, $\mathsf{b}_i \in \{0, 1\}$, $x_i \in \mathcal{V}^{\mathbb{R}}$ for all $i = 1, \ldots, m$, and $\mathsf{c} \in \mathbb{R}$.

Formulas of the *generic constraint logic* (GCL) are simply propositional formulas over atomic constraints.

$$\varphi \ ::= C \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi$$

where C is one of the constraints above. It should be clear that, with Boolean operators at hand, many other constraints become definable, for instance $x = \mathsf{c} := x \leq \mathsf{c} \wedge -x \leq -\mathsf{c}$, $x < \mathsf{c} := x \leq (\mathsf{c}-1)$ in case of an integer constraint, etc.

We write $|\varphi|$ to denote the size of φ measured in terms of occurrences of logical, relational and arithmetic symbols as well as size of a representation for the constants, for instance encoded in binary.

A variable *assignment* is a $\vartheta = (\vartheta^{\mathbb{Z}}, \vartheta^{\mathbb{R}})$ s.t. $\vartheta^{\mathcal{X}} : \mathcal{V}^{\mathcal{X}} \to \mathcal{X}$ for $\mathcal{X} \in \{\mathbb{Z}, \mathbb{R}\}$. Hence, it assigns integer, resp. real values to the integer, resp. real variables. The *truth* value of a formula φ under an assignment ϑ is defined straight-forwardly by evaluating the constraints and subformulas under the usual rules for $<$, $\leq$, $+$, $-$, $\wedge$, $\vee$, and $\neg$. We write $\vartheta \models \varphi$ to state that φ evaluates to *true* under the assignment ϑ.

A formula is *satisfiable* if there is a ϑ s.t. $\vartheta \models \varphi$. Clearly, such a ϑ only needs to be defined on the variables that actually occur in φ which allows us to assume that these assignments have finite domains. Two formulas are equivalent, written $\varphi \equiv \psi$, if for all ϑ we have $\vartheta \models \varphi$ iff $\vartheta \models \psi$.

A formula is *positive* it it does not contain any occurrences of the negation operator $\neg$. Note that this is stronger than the well-known concept of positive normal form which still would allow negation operators. The following is easy to prove using the duality between $\leq$ and $<$.

Lemma 1. *For every φ there is a positive ψ s.t. $\varphi \equiv \psi$ and $|\psi| \leq 2|\varphi|$.*

Theorem 3. *The satisfiability problem for* GCL *is NP-complete.*

Proof. NP-hardness is a simple consequence of the fact that integer variables can be restricted to the domain $\{0, 1\}$ using the available constraints and thus model Boolean variables. It is therefore possible to embed the NP-hard satisfiability problem for propositional logic [11] into the satisfiability problem for GCL.

For the upper bound we describe a nondeterministic algorithm. Let φ be a GCL formula. According to Lemma 1 we can assume φ to be positive. First guess a truth value for each atomic constraint in it. Clearly, there are at most $|\varphi|$ many, and the truth values can be propagated up in the formula in polynomial time to see whether the result is true. Next we need to decide whether there are variable assignments that fulfil the atomic constraints which are guessed to be true. Note that the integer constraints and the real constraints are independent of each other because they do not share any variables. Hence, they can be dealt with separately.

Solving a set of integer constraints of the form $\mathsf{b} \cdot x \leq \mathsf{c}$ can be done in polynomial time using interval arithmetic. Note that the solution to each such constraint is given by an interval $(-\infty, \mathsf{c}]$ if $\mathsf{b} = 1$, and $[\mathsf{c}, \infty)$ if $\mathsf{b} = -1$. Computing intersections of such constraints is easy.

In order to deal with real-valued constraints we use the linear programming optimisation problem. First of all, we add a new variable y and replace every strict inequality $\sum_{i=1}^{m} \mathsf{b}_i \cdot x_i < \mathsf{c}$ by $y + \sum_{i=1}^{m} \mathsf{b}_i \cdot x_i \leq \mathsf{c}$. This creates a linear program with cost function y, i.e. we are looking for a solution to these constraints that maximises the value of y. In order to ensure that there is a maximal value we also add a constraint like $y \leq 1$. It is known that the linear programming optimisation problem can be solved in deterministic polynomial time when the coefficients are also taken into account as part of the input [16]. Finally, the result of the optimisation problem needs to be translated back into a result of the original system of constraints. If the system is not solvable then so is the original one. If the maximal feasible value for y is at most 0 then the original system is also not solvable. If the maximal feasible value for y is strictly positive, then there is also a solution to the original system. □

4.2 The Incremental Encoding

We fix a TA $\mathcal{A} = (L^{\mathcal{A}}, L_0^{\mathcal{A}}, L_F^{\mathcal{A}}, \Sigma, X^{\mathcal{A}}, E^{\mathcal{A}})$, a DTA $\mathcal{B} = (L^{\mathcal{B}}, q_0^{\mathcal{B}}, L_F^{\mathcal{B}}, \Sigma, X^{\mathcal{B}}, E^{\mathcal{B}})$ and a natural number k for the remainder of this section. We will define a formula $\varphi_k^{\mathcal{A},\mathcal{B}}$ that is satisfiable iff $L(\mathcal{A}) \not\subseteq_k L(\mathcal{B})$. Moreover, its size will be linear in the sizes of $|\mathcal{A}|$, $|\mathcal{B}|$, and in k. Note that this is the value of k, not the number of bits needed to represent it.

In order to make the presentation of $\varphi_k^{\mathcal{A},\mathcal{B}}$ as intuitive as possible, we divide it into several parts and present it statically first. At the end we will discuss how to make it incremental in the sense that the formula $\varphi_{k+1}^{\mathcal{A},\mathcal{B}}$ can be obtained from $\varphi_k^{\mathcal{A},\mathcal{B}}$ by deleting and adding single components.

The first part states that the witnessing word is well-formed. It uses natural number variables $a_0, \ldots, a_{k-1}$ to encode the events in a witness, as well as real-valued variables $t_0, \ldots, t_{k-1}$ for the time delays of such a witness. W.l.o.g. we assume the alphabet to be $\{1, \ldots, |\Sigma|\}$. Furthermore, remember that the formula to be built should state that there is a word of length *at most* k with some property. In order to capture words of length $< k$ as well, we take an additional value, say 0, and represent such a shorter timed word w by $w(0, t_1) \ldots (0, t_m)$ where $m = k - |w|$ and the t_i are arbitrary values. The following formula then states well-formedness of such a timed word.

$$\mathit{word}_k \ := \ \big(\bigwedge_{i=0}^{k-1} 0 \leq t_i\big) \ \wedge \ \big(\bigwedge_{i=0}^{k-1} 0 \leq a_i \wedge a_i < |\Sigma|\big) \ \wedge \ \big(\bigwedge_{i=0}^{k-2} a_i = 0 \ \rightarrow \ a_{i+1} = 0\big)$$

Next we formalise that the timed word w represented by values for those variables is accepted by $\mathcal{A}$. We use $k+1$ integer variables $\ell_0, \ldots, \ell_k$ to represent the locations in an accepting run of $\mathcal{A}$ on w. W.l.o.g. we can assume that $L^{\mathcal{A}} = \{0, \ldots, |L^{\mathcal{A}}| - 1\}$. We also use $(k+1) \cdot |X^{\mathcal{A}}|$ many real-valued variables v_i^c where $c \in X$ and $0 \leq i \leq k\}$, in order to represent the values of the clocks in each state of this accepting run. We first state that the run is well-formed in the sense that each pair of adjacent states is connected by a transition that is possible in $\mathcal{A}$ unless the event symbol at that position is 0.

$$\mathit{run}_k^{\mathcal{A}} \ := \ \bigwedge_{i=0}^{k-1} \bigwedge_{\mathsf{q} \in L^{\mathcal{A}}} \ell_i = \mathsf{q} \wedge \neg(a_i = 0) \ \rightarrow \bigvee_{(\mathsf{q}, g, \mathsf{a}, R, \mathsf{q}') \in E^{\mathcal{A}}} \mathit{sat}_i(g) \ \wedge \ a_i = \mathsf{a} \ \wedge \ \mathit{progress}_i(R) \ \wedge \ \ell_{i+1} = \mathsf{q}'$$

where $sat_i(g)$ states that the guard g is satisfied by the clock valuation after the i-th delay has passed. It is defined by induction on the structure of g:

$$\begin{array}{rclrcl} sat_i(x \leq \mathsf{c}) & := & v_i^x + t_i \leq \mathsf{c} & sat_i(\neg g) & := & \neg sat_i(g) \\ sat_i(x < \mathsf{c}) & := & v_i^x + t_i < \mathsf{c} & sat_i(g_1 \wedge g_2) & := & sat_i(g_1) \wedge sat_i(g_2) \end{array}$$

Furthermore, $progress_i(R)$ formalises the progression of time. In particular, the clock valuation at the next moment results from delaying at the current moment and resetting clocks afterwards.

$$progress_i(R) \;:=\; \big(\bigwedge_{x \in R} v_{i+1}^x = 0\big) \;\wedge\; \big(\bigwedge_{x \in X^{\mathcal{A}} \setminus R} v_{i+1}^x = v_i^x + t_i\big)$$

Finally, we need to say that the run is accepting, i.e. starts in an initial location with all the clocks set to 0 at that moment. A final location needs to be reached at that point when the next event symbol either does not exist anymore (i.e. the k-the location in the run) or is 0 (for words of length $< k$).

$$\begin{array}{rcl} acc_k^{\mathcal{A}} & := & \big(\bigwedge_{x \in X^{\mathcal{A}}} v_0^x = 0\big) \;\wedge\; \big(\bigvee_{\mathsf{q} \in L_0^{\mathcal{A}}} \ell_0 = \mathsf{q}\big) \;\wedge\; \big(\bigwedge_{i=0}^{k-1} a_i = 0 \;\rightarrow\; \bigvee_{\mathsf{q} \in L_F^{\mathcal{A}}} \ell_i = \mathsf{q}\big) \;\wedge \\ & & \big(\neg(a_{k-1} = 0) \;\rightarrow\; \bigvee_{\mathsf{q} \in L_F^{\mathcal{A}}} \ell_k = \mathsf{q}\big) \end{array}$$

Next we need to state that $\mathcal{B}$ does not accept the timed word given by the values for the a_i and t_i variables. In general, this is a universal statement over all runs of the automaton but, since it is assumed to be deterministic and complete, it can equivalently be rephrased as "there is a run that is not accepting". Hence, the constraints $run_k^{\mathcal{B}}$ are formed exactly in the same way as for $run_k^{\mathcal{A}}$ but using fresh variables $\ell_0', \ldots, \ell_k'$ for the locations in the run of $\mathcal{B}$. W.l.o.g. we can assume $X^{\mathcal{A}} \cap X^{\mathcal{B}} = \emptyset$, i.e. the two automata use different clocks. We then have variables v_i^x as above for every clock $x \in X^{\mathcal{B}}$ and every $i = 0, \ldots, k$.

We also define a constraint $rej_k^{\mathcal{B}}$ similar to $acc_k^{\mathcal{A}}$ now stating that the run of $\mathcal{B}$ given by the valuations of the ℓ_i' and v_i^x variables is rejecting.

$$rej_k^{\mathcal{B}} \;:=\; \big(\bigwedge_{x \in X^{\mathcal{B}}} v_0^x = 0\big) \wedge \ell_0 = \mathsf{q}_0^{\mathcal{B}} \wedge \big(\bigwedge_{i=0}^{k-1} \bigwedge_{\mathsf{q} \in L_F^{\mathcal{A}}} a_i = 0 \rightarrow \ell_i' \neq \mathsf{q}\big)$$

Finally, we define $\varphi_k^{\mathcal{A},\mathcal{B}} \;:=\; word_k \wedge run_k^{\mathcal{A}} \wedge acc_k^{\mathcal{A}} \wedge run_k^{\mathcal{B}} \wedge rej_k^{\mathcal{B}}$. There are two important aspects to note about $\varphi_k^{\mathcal{A},\mathcal{B}}$: it is small, i.e. linear in its three parameters, and it characterises the bounded TA-DTA inclusion problem.

Proposition 3. $|\varphi_k^{\mathcal{A},\mathcal{B}}| = \mathcal{O}(k \cdot (|\mathcal{A}| + |\mathcal{B}|))$.

Theorem 4. $\varphi_k^{\mathcal{A},\mathcal{B}}$ *is satisfiable iff* $L(\mathcal{A}) \not\subseteq_k L(\mathcal{B})$.

Proof. "$\Leftarrow$" Suppose $L(\mathcal{A}) \not\subseteq_k L(\mathcal{B})$. Then there is a $w = (\mathsf{a}_0, \mathsf{t}_0), \ldots, (\mathsf{a}_{n-1}, \mathsf{t}_{n-1}) \in T\Sigma^*$ s.t. $n \leq k$ and $w \in L(\mathcal{A})$ and $w \notin L(\mathcal{B})$. Then there is an accepting run $\langle q_0, v_0\rangle, \ldots, \langle q_n, v_n\rangle$ of $\mathcal{A}$ on w. Furthermore, the unique run $\langle q'_0, v'_0\rangle, \ldots, \langle q'_n, v'_n\rangle$ of $\mathcal{B}$ on w is not accepting, i.e. it does not end in a final location.

It is now routine to check that the variable assignment $\vartheta = (\vartheta^{\mathbb{Z}}, \vartheta^{\mathbb{R}})$ satisfies $\varphi_k^{\mathcal{A},\mathcal{B}}$, where

$$\begin{array}{rclrcl}
\vartheta^{\mathbb{Z}}(a_i) & = & \begin{cases} \mathsf{a}_i & \text{, if } i < n \\ 0 & \text{, if } n \leq i < k \end{cases} & \vartheta^{\mathbb{R}}(t_i) & = & \begin{cases} \mathsf{t}_i & \text{, if } i < n \\ 0 & \text{, if } n \leq i < k \end{cases} \\
\vartheta^{\mathbb{Z}}(\ell_i) & = & \begin{cases} \mathsf{q}_i & \text{, if } i \leq n \\ \mathsf{q}_{n-1} & \text{, if } n < i \leq k \end{cases} & \vartheta^{\mathbb{R}}(v_i^x) & = & \begin{cases} v_i(x) & \text{, if } i \leq n \\ 0 & \text{, otherwise} \end{cases} \\
\vartheta^{\mathbb{Z}}(\ell'_i) & = & \begin{cases} \mathsf{q}'_i & \text{, if } i \leq n \\ \mathsf{q}'_{n-1} & \text{, if } n < i \leq k \end{cases} & & &
\end{array}$$

"$\Rightarrow$" Suppose $\varphi_k^{\mathcal{A},\mathcal{B}}$ has a satisfying variable assignment $\vartheta = (\vartheta^{\mathbb{Z}}, \vartheta^{\mathbb{R}})$. The $\vartheta^{\mathbb{Z}}$-values of $a_0, \ldots, a_{k-1}$ and the $\vartheta^{\mathbb{R}}$-values of $t_0, \ldots, t_{k-1}$ encode a timed word over the alphabet $\Sigma \cup \{0\}$. Because of $word_k$ this is in fact a timed word in $T(\Sigma^* 0^*)$. Hence, there is an $n \leq k$, s.t. $w' := (\vartheta^{\mathbb{Z}}(a_0), \vartheta^{\mathbb{R}}(t_0)), \ldots, (\vartheta^{\mathbb{Z}}(a_{n-1}), \vartheta^{\mathbb{R}}(t_{n-1})) \in T\Sigma^*$. Furthermore, consider the sequence $\langle \vartheta^{\mathbb{Z}}(\ell_0), v_0\rangle, \ldots, \langle \vartheta^{\mathbb{Z}}(\ell_n), v_n\rangle$ where $v_i(x) = \vartheta^{\mathbb{R}}(v_i^x)$ for every $i = 0, \ldots, n$ and every $x \in X^{\mathcal{A}}$. Because of $run_k^{\mathcal{A}}$ this forms a run of $\mathcal{A}$ on w' which can be extended to a run on w by simply repeating the last pair in this sequence $k - n$ times. Because of $acc_k^{\mathcal{A}}$ this run is accepting; it starts with all clocks set to 0 and ends in a final location.

In the same way one can see that $run_k^{\mathcal{B}}$ and $rej_k^{\mathcal{B}}$ enforce the unique run of $\mathcal{B}$ on w to be rejecting. Hence, we have $L(\mathcal{A}) \not\subseteq_k L(\mathcal{B})$. □

It should be clear that similar formulas characterising the bounded TA emptiness or the bounded DTA universality problem can easily be defined as well. In fact, it is not necessary to carry out the cnstructions sketched for Prop. 1. Instead, for bounded emptiness it suffices to remove the constraints involving $\mathcal{B}$ in $\varphi_k^{\mathcal{A},\mathcal{B}}$; for bounded universality it suffices to remove those involving $\mathcal{A}$.

Also note that the encoding can easily be made *incremental* in the sense that the formula $\varphi_{k+1}^{\mathcal{A},\mathcal{B}}$ can be obtained by certain operations on $\varphi_k^{\mathcal{A},\mathcal{B}}$. In detail:

1. Add the conjuncts $0 \leq t_k$, $0 \leq a_k$, $a_k \leq |\Sigma|$, $a_{k-1} = 0 \rightarrow a_k \rightarrow 0$ to $word_k$.
2. Extend the conjunctions in $run_k^{\mathcal{A}}$ and $run_k^{\mathcal{B}}$ by one to the range $i = 0, \ldots, k$.
3. Remove the last conjunct from $acc_k^{\mathcal{A}}$, extend the conjunction before that to the range $i = 0, \ldots, k$, and add the conjunct $\neg(a_k = 0) \rightarrow \bigvee_{\mathsf{q} \in L_F^{\mathcal{A}}} \ell_{k+1} = \mathsf{q}$.

Finally, the translation gives us an upper complexity bound matching the lower bound in Thm. 2. It follows immediately from Thm. 4 and Thm. 3.

Corollary 1. nEMPTY$^{\leq}$ *and* ndINCL$^{\leq}$ *are NP-complete for a unarily encoded bounding parameter k. The lower bound requires at least two clocks. Also,* dUNIV$^{\leq}$ *is in co-NP.*

4.3 Translations into Other Formalisms

The extension of the propositional logic proposed above is extensive enough for our purpose here, which is the encoding of bounded representations of the three decidability problems under consideration. A natural question that arises is how this can help to solve these problems in practice.

We refer to the SMT-LIB [7] framework, a standardisation for propositional and predicate logics over various domains. It contains well-documented input languages, including theories of difference logic over integers and reals and their combinations. It is not hard to see that the propositional logic defined above can esily be embedded into some of these formalisms, for instance the combination of `QF_IDL` (difference logic over integers) and `QF_RDL` (difference logic over the reals).

Furthermore, most SMT solvers nowadays comply with the SMT-LIB standards in the sense that it is clearly stated which theories they handle.

5 Implementation and Experimental Results

The approach described above is realised in a prototypical implementation done in OCaml. It creates constraints according to some pre-defined families of benchmarks, and uses the SMT solver Z3 in order to solve the bounded emptiness, inclusion or universality problem as defined above in an incremental way.

All tests reported here were run on a compute server equipped with 16 Intel Xeon cores at 1.87GHz and 256GB main memory. Note that neither the reduction nor the SMT solver support parallelism, thus each test only occupies one core.

Here we report on three benchmarks testing different aspects of this approach, all phrased as some series of emptiness problems.

Fischer's Mutex Protocol. The first benchmarking family models Fischer's protocol for mutual exclusion between n processes communicating via one shared variable as one timed-automaton. Note that the state-spaces of the TA are exponential in n. Their languages consist of all runs of the asynchronous product of these n processes which end in a state that has at least two processes in the critical section. Thus, this is a classical safety verification problem. In this

Table 1. Experimental analysis of emptiness for timed automata

Fischer's Mutex protocol

n	size	create	solve
2	4238	0.01	0.02
3	33750	0.05	0.93
4	230760	0.20	20.86
5	1433948	1.16	6511.16

Diagonal Constraints

s	length	size
1	1141	197422
5	1040	179096
10	1160	198972
20	1299	224922
50	1499	259522
100	1699	294122

Exponential Witnesses

n	size	create	solve	c
2	1136	0.01	0.00	5
3	3230	0.02	0.02	8
4	8415	0.05	0.27	14
6	37121	0.32	42.77	38
8	179827	89.54	7050.78	134

model, the delay times are set such that mutual exclusion is not guaranteed for otherwise the languages would be empty.

Table 1 presents experimental data showing the size of the resulting formula in number of logical operators in it, the time it takes to create it and the time it takes to solve it in seconds. This benchmark is created in order to stress the point that SMT solvers can check relatively large formulas for satisfiability.

Diagonal Constraints. There is an example of an 8-state TA using diagonal constraints in [9]. Its language is empty. The example shows that having diagonal constraints in the timed automaton makes the classical forward analysis approach, implemented in most timed automata verification tools like KRONOS and UPPAAL, unsound.

We use this benchmark to demonstrate that the approach in this paper can easily deal with diagonal constraints, but also to examine the effect of varying the step width s in which larger and larger ranges of lengths of timed words are considered. Thus, a value of $s = 7$ for instance means checking for witnesses of length $\{0, \ldots, 6\}$ first, then of length $\{7, \ldots, 13\}$, etc.

Table 1 presents the maximal range length that could be examined within a run time of one hour, as well as the size of the formula at the end. Thus, the implementation showed for instance within one hour, that the TA under consideration does not accept any word of length at most 1699. The interpretation of the data is: larger incremental steps can save some time.

Exponential Witnesses. As a last benchmark we consider a family of TA with $n + 1$ states s.t. their language is non-empty but the shortest word that they accept is of length $2^n - 1 + n$. This can be enforced with a trick that is similar to the one used in the NP-hardness proof in Thm. 2. However, a naïve approach would cause a quadratic number of transitions. Here we use a slightly randomised model of such TA which only needs a linear number of transition.

Table 1 shows size of formula, creation and solving time in seconds as well as number c of calls to the SMT solver before a witness is being found. The step width for ranges in the incremental analysis is fixed, thus this number grows exponentially. Note that the reported solving time is the accumulated time of c calls. This benchmark shows that the SMT solver can still find relatively long witnessing words, here for instance of length $263 = 2^8 - 1 + 8$. Note that the numbers for formula sizes in the table clearly do not grow linearly. This is because they represent formulas for exponentially differing ranges of word length, namely those of the shortest words witnessing non-emptiness.

6 Further Work

There is a vast possibility for follow-up work. For starters, the encoding should be extended to richer models, for instance TA with state invariants which is very easy. It is also worth considering asynchronous networks of TA. It is easy to see that the emptiness problem can be encoded in the same way, but the two other problems may not be capturable as concurrency introduces nondeterminism. Then there is the question of extending this to TA on infinite timed words.

There are two more extensions which show the vast potential that the use of SMT solvers has as opposed to using SAT solvers: the latter are restricted to problems in NP, the former are hardly restricted at all. Thus, far richer clock constraint languages (for instance with all arithmetic operations) can easily be handled by SMT solvers. At last, a very promising extension is the consideration of richer logical (rather than just arithmetic) formalisms on the SMT side, for instance logics with quantification. It remains to be seen which problems on timed automata can also be encoded this way. Note that this is not restricted to decidable problems since there are SMT logics which are undecidable in general. Still, using SMT solvers for these problems may offer a practically viable approach which may just not be complete or not always terminating.

References

1. `http://research.microsoft.com/en-us/um/redmond/projects/z3/`
2. Abdulla, P.A., Deneux, J., Ouaknine, J., Quaas, K., Worrell, J.: Universality Analysis for One-Clock Timed Automata. Fundamenta Informaticae, 89 (2008)
3. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theo. Comp. Sci. (1994)
4. Alur, R., Madhusudan, P.: Decision Problems for Timed Automata: A Survey. In: SFM School (2004)
5. Audemard, G., Bozzano, M., Cimatti, A., Sebastiani, R.: Verifying industrial hybrid systems with mathsat. ENTCS 119(2) (2005)
6. Audemard, G., Cimatti, A., Kornilowicz, A., Sebastiani, R.: Bounded model checking for timed systems. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE 2002. LNCS, vol. 2529, Springer, Heidelberg (2002)
7. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. Technical report (2010), `http://www.SMT-LIB.org`
8. Bérard, B., Petit, A., Diekert, V., Gastin, P.: Characterization of the expressive power of silent transitions in timed automata. Fundam. Inform. 36(2-3) (1998)
9. Bouyer, P.: Untameable timed automata! In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 620–631. Springer, Heidelberg (2003)
10. Bouyer, P., Laroussinie, F., Reynier, P.-A.: Diagonal constraints in timed automata: Forward analysis of timed systems. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 112–126. Springer, Heidelberg (2005)
11. Cook, S.A.: The complexity of theorem-proving procedures. In: 3rd Annual ACM Symposium on Theory of Computing (STOC), pp. 151–158 (1971)
12. de Moura, L.M., Rueß, H., Sorea, M.: Bounded model checking and induction: From refutation to verification (extended abstract, category a). In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 14–26. Springer, Heidelberg (2003)
13. Fränzle, M., Herde, C.: HySAT: An efficient proof engine for bounded model checking of hybrid systems. Formal Methods in System Design 30(3) (2007)
14. Henzinger, T.A., Manna, Z., Pnueli, A.: What good are digital clocks? In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, Springer, Heidelberg (1992)
15. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103 (1972)

16. Khachiyan, L.G.: A polynomial algorithm in linear programming. Doklady Akademiia Nauk SSSR, 224 (1979)
17. Niebert, P., Mahfoudh, M., Asarin, E., Bozga, M., Maler, O., Jain, N.: Verification of timed automata via satisfiability checking. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 225–243. Springer, Heidelberg (2002)
18. Ouaknine, J., Worrell, J.: Revisiting digitization, robustness, and decidability for timed automata. In: LICS (2003)
19. Ouaknine, J., Worrell, J.: On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap. In: LICS (2004)
20. Strichman, O.: Pruning techniques for the SAT-based bounded model checking problem. In: Margaria, T., Melham, T.F. (eds.) CHARME 2001. LNCS, vol. 2144, pp. 58–70. Springer, Heidelberg (2001)
21. Zbrzezny, A.: SAT-based Reachability Checking for Timed Automata with Diagonal Constraints. Fundam. Inform. 67(1-3), 303–322 (2005)

Segmented State Space Traversal for Conformance Testing of Cyber-Physical Systems

Matthias Woehrle[1], Kai Lampka[2], and Lothar Thiele[2]

[1] Embedded Software Group, Delft University of Technology, The Netherlands
m.woehrle@tudelft.nl
[2] Computer Engineering and Networks Lab, ETH Zurich, Switzerland
{lampka,thiele}@tik.ethz.ch

Abstract. Quantitative conformance testing of cyber-physical system (CPS) exploits time series of measurements, such as temperature or energy, for validating the correctness of deployed systems. This paper presents the foundations of *segmented state space traversal* in the setting of quantitative conformance testing of a CPS. It is demonstrated how this strategy together with domain-specific adaptations remedies state space explosion inherent to formal (state-based) verification. The presented contributions improve the scalability of quantitative conformance testing of a CPS and is demonstrated with a case study.

1 Introduction

Motivation: Cyber-physical systems (CPS) operate on the edge between the physical world and its continuous quantities, and the discrete world of computing devices. It is this complexity that makes the design and implementation of CPS difficult and error-prone; tragic loss and costly damages due to system flaws were encountered in the past. Hence correctness of system design is not only desired, but a requirement. In the cyber-physical setting correctness does not only encompass algorithmic and functional aspects, but also includes extra-functional properties, e. g. temperature levels, power consumption and timing. However, verifying the correctness of state-based system models commonly requires the unfolding of all possible behaviors. Due to the notorious state space explosion problem this may fail in practice; traversal and storage of exponentially many states induces a non-tolerable memory and computation time overhead. As an alternative to verification, formal frameworks for testing have shown their usefulness, and remain the predominantly choice of industry when it comes to validating the correctness of system's implementation. In previous work [12,13] we developed a Timed Automata (TA) based technique that exploits time series of measurements for uncovering implementation flaws of CPS. The proposed framework is implemented on top of a standard timed model checker and therefore relies on the efficiency of the state space traversal scheme of the employed model checker. However, the quantitative conformance testing as proposed by

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 193–208, 2011.

us, together with the characteristics of CPS to be tested yields a problem formalization, whose structure allows us to use problem-specific improvements in the state space traversal. These improvements are the focus of this paper.

Related Work: In previous works we have have proposed a framework for quantitative conformance testing of CPS [12,13]. In a nutshell, our TA-based technique works as follows: (a) from a running system one obtains a timed series of measurement values. (b) The data points of the measured trace are aggregated and translated into a (linear) observer TM. (c) Joint execution of the trace model TM and a TA-based model of the system allows for checking the reachability of the terminal location of the trace model TM. If this is possible, we infer that the measurements conform to the expected behavior of the system. This scheme clearly differs from real-time testing with TA-based tester as introduced in a series of publications, e. g. [5,9], where one derives a tester from a TA-based system model and applies it to the system's real-world implementation.

As pointed out above, the here presented conformance testing depends on a reachability check. This requires a state space traversal until either the terminal location of TM is contained in a state labeling or until all system states have been visited. For a state space exploration routine to terminate it is essential that one does not re-explore states that have already been under consideration.

Christensen et al. [6] present the sweep-line method where one removes states from the memory as soon as they are known to be unreachable for all possible evolutions of the system (model), i. e. once it is known that the system can not loop back to them. In turn this reduces the memory overhead as imposed by the storage of states that are not needed to be stored since the state space traversal will never visit them again. A similar line of thought can be exploited for quantitative conformance testing where a TA-based system model is tested against a measured trace. When carrying out the state space traversal in a breadth-first search (bfs) manner, previously generated states can be dropped as soon as the next reading from the measured trace is processed. This technique, which is due to the linear character of any measured trace, has been implemented in the Uppaal-based tool TRON [8]. When executing a conformance test TRON generates the trajectory of the modeled system, w. r. t. a set of possible states, and w. r. t. the time stamp of the next data point of the measured trace. Now, the newly generated states are filtered w. r. t. the current reading of the measured trace, and all non conformant trajectories, as well as all previously generated states are discarded, or dropped respectively. This procedure that starts with the initial system configuration is repeated until the end of the measured trace is reached or until all possible system trajectories have been discarded. A drawback is that this procedure requires to store and explore all successor states which are conformant to the current reading from the measured trace. Hence, it might, nevertheless, suffer from the state space explosion problem, especially when dealing with system models showing a high-degree of non-determinism; this issue has been studied in [13] and is also shown in Sec. 2.3. In this paper we follow a different approach: contrary to TRON and its bfs-based state space traversal we follow a depth-first search (dfs) and segmented scheme and thereby

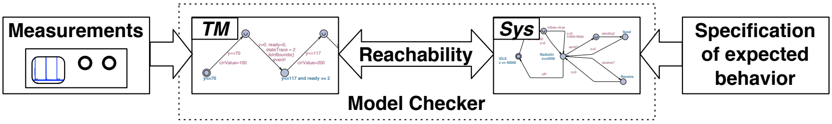


Fig. 1. Conformance testing with physical quantities: the scheme

trading memory with runtime. This is useful as for complex models conformance testing tends to be infeasible due to the limited availability of memory.

In recent work McMillan [11] developed the lazy annotation method. This scheme aims to produce a witness or counter-example for a given system model and w. r. t. a safety property, i. e. a property that can be verified by executing a state space traversal, or a reachability analysis respectively. At its core the proposed techniques steers the traversal on a system's abstract computation tree by inferring Floyd/Hoare style annotations. These annotations prevent the traversal to repetitively branch into directions that have already been identified as unsuccessful w. r. t. the property under consideration. Our approach, the segmented state space traversal presented in this paper, can be seen as an extension of McMillan's lazy annotation approach to the setting of TA. This is because the unfolding of a TA's underlying zone graph until a dedicated goal location (in this work the terminal location of the trace automaton) is marked active resembles the unfolding of the computation tree. However, whereas McMillan employs Floyd/Hoare style annotations the method of this paper make use of symbolic clock evaluations, i. e. clock zones, for preventing the traversal to branch into directions that have previously been identified as unsuccessful. Due to the linear nature of the trace automaton and re-storage of clock zones, our technique can be organized in independent runs of the model checker, whereas in case of McMillan's lazy annotation re-exploration of known system configurations must be prevented.

Contribution: In Sec. 3 this paper develops a segmented scheme for organizing the state space traversal, in order to keep the memory requirements below a threshold, and thereby achieving scalability of quantitative conformance testing to real-world CPS. Such a real-world case study is presented in Sec. 4, where we test the conformance of concurrently obtained power measurements of two communicating wireless sensor nodes and an abstract system model. This test is not feasible without our proposed innovations.

2 Background Theory

2.1 Preliminaries

For conformance testing we exploit TA [1] with variables and location invariants. A set of such TA can be composed into a network of cooperating TA, where the notation $\mathcal{A}||\mathcal{B}$ refers to the parallel execution of (component) TA $\mathcal{A}$ and (component) TA $\mathcal{B}$. In the following we simply speak of TA, but in fact are referring to networks of TA. Furthermore we informally clarify now only few, but relevant details; cf. [1,14,4,3] for further information.

An edge traversal in a TA may change its (active) locations, may reset a set of clocks and may update the set of variables. This operational character is referred to by edge executions. Due to the dense time model where delays of edge executions can be sampled from intervals in $\mathbb{R}^+$, a TA may show infinitely many different behaviors. However, by exploiting the notion of clock regions the infinite behavior of a TA A can be captured by a finite graph $\mathcal{TS}^A$. The nodes of this graph, commonly denoted as system configurations or states refer to clock regions, rather than individual clock evaluations; for details cf. [1,14]. In the reminder of this paper we are also dealing with non-zeno systems only. Thereby we exclude TA where the subsequent execution of infinitely many edges is possible without the time progressing.

A system configuration or state of a network of TA is a tuple $\boldsymbol{s} = (\boldsymbol{l}, \mathbf{c}, \boldsymbol{v})$ where

- $\boldsymbol{l} = (l_0, ..., l_n)$ is a vector of location identifier. Its i'th component refers to the active location of the i'th component TA; the active location of a TA is the location the TA currently resides in.
- $\mathbf{c}$ is a set of clock constraints defining a region (or zone) and
- $\boldsymbol{v}$ is the vector of values currently held by the variables of the TA.

The initial state $\boldsymbol{s}^0$ of a network of TA is defined by initial locations of the component TA, clocks all set to 0 and by an initial assignment for all variables.

A path in the state graph underlying a TA is a sequence of transitions (edges of the state graph) and states (nodes of the state graph) strictly alternating (cf. Fig. 3). A transition of the state graph either refers to the execution of an edge of the TA or to an abstract delay transition, where the latter represents the progress of time. With $\pi(\boldsymbol{s}^0)$ we refer to a path starting at the initial system configuration and $\pi(\boldsymbol{s}^0)[i]$ addresses the i'th state visited on path $\pi(\boldsymbol{s}^0)$. With Π^A we refer to the set of all paths constructible from A. In the following we will also use the notation $\boldsymbol{s}^j_\alpha$. This addresses a state $\boldsymbol{s}$, indexed by j and containing location identifier (or label) α.

As demonstrated below, quantitative conformance testing as proposed by us can be mapped to finding the answer to a reachability problem of a network of TA and w. r. t. a location identifier. Hence this paper is solely concerned with finite terminal paths that start in state $\boldsymbol{s}^0$ and end in some state $\boldsymbol{s}^k$, opposed to liveness properties that refer to non-terminal paths.

A location identifier α is denoted reachable in a TA A iff there exists a path (fragment) $\pi(\boldsymbol{s}^0)$ that ends in a state $\boldsymbol{s}^j_\alpha$, with j as some state index and we write:

$$A \models \texttt{E<>}\, \alpha \Leftrightarrow \exists \pi(\boldsymbol{s}^0) : \pi(\boldsymbol{s}^0)[i] = \boldsymbol{s}^j_\alpha \text{ for some } i, j \in \mathbb{N}_0 \quad (1)$$

In the above equation $\models$ is the satisfaction relation defined on TA and the set of CTL formulae.

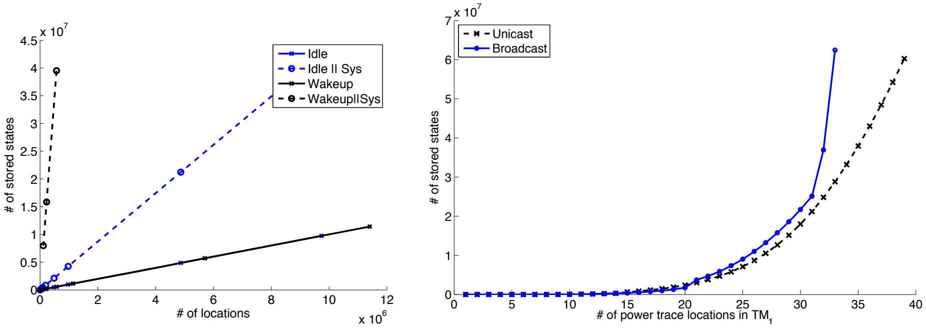
2.2 Quantitative Conformance Testing

Having formal models for both measurement trace (TA TM) and (expected) system behavior (TA Sys), the state graph of composite $Sys||TM$ can be generated. The conformance test is considered positive iff the terminal location of TM is reachable. In the following, we briefly detail on these aspects.

The System Model $\boldsymbol{Sys}$: The locations of Sys are annotated with clock and variable invariants. This results in a description of the system w. r. t. timing and the behavior of the physical quantity to be exploited for the conformance test, e. g. power consumption. For dealing with deviations in the physical quantities we partition its continuous value space into finitely many intervals (see also discussion on greatest common intervals following below). The interval associated with the current system state is defined by a pair of (global) variables h_{low}, h_{up}. For system models that are networks of timed automata, the overall bounds, i. e. the values of h_{low} and h_{up}, can be obtained by aggregating the bounds of the component TA accordingly, where aggregation and updating takes place upon edge execution.

Trace Model $\boldsymbol{TM}$: The finite trace of measurements is translated into a TA TM that features a single clock for modeling the individual holding times of the measurements. The individual measurement values are stored as a variable that we denote p. This yields a finite sequence of locations and edges where the terminal location is equipped with the label `final`. For obtaining a compact trace model we advocate the usage of intervals w. r. t. the measured physical quantity, rather than using the exact, measured values. This way we can compress millions of data points into a processable number of locations in TM [12]. The intervals are defined by all possible combinations of the upper and the lower values to be assigned to h_{up}, respectively h_{low} of Sys. These combinations, e. g. by addition for power consumption, feature the a priori definition of a set of disjoint greatest common intervals (GCI) for the construction of TM. In the measured trace we replace each measurement value with the mean of the corresponding GCI. Adjacent, equal measurement values are modeled by a single location in TM and by adapting the location-holding times accordingly. This yields a quotient TA that exhibits an equivalent behavior w. r. t. the system model and w. r. t. the GCI-membership of a measurement point [13]. In the following, TM refers to such a quotient TA.

Conformance Testing is performed by querying the reachability of location `final` of the composite $Sys||TM$. When jointly executing Sys and TM, the model checker may execute enabled edges of Sys as long as for the generated system states invariant $h_{low} \leq p \leq h_{up}$ holds. On the other hand variable p is updated when edges of TM are executed. However, updates of p may invalidate this invariant. This is resolved by using intermediate update location in the component TA of Sys, i. e. locations where time does not pass and where the above invariant w. r. t. the modeled quantity is absent. Once an update location is marked active in a state, yet for all successor states the invariant would be violated, the state space traversal yields a deadlock. This behavior is enforced by Sys as all successor locations of an update location require once again the validity of invariant $h_{low} \leq p \leq h_{up}$ when being entered and marked active. On



a) State space size as a function of trace model length for one sensor node

b) State space size as a function of trace model length for two sensor nodes

Fig. 2. State space size depending on trace model length and model complexity

the other hand the state space traversal yields no deadlock if there is at least one successor location where the above invariant holds. Hence, this may lead to reaching a state where location `final` is marked active. In this case a path has been generated that witnesses the conformance of measured trace and system model Sys.

2.3 Scalability, an Open Issue

Now we study the feasibility of quantitative conformance testing.

Sensitivity w. r. t. the Length of the Time Series of Measurements: In the case study of Sec. 4 we analyze wireless sensor nodes, where we exploit power measurements for conformance testing. In a series of experiments we employed a single sensor node and two kinds of trace models, $Idle$ and $Wakeup$ (cf. Fig. 2). $Idle$ is an artificial trace model, with the sensor node permanently residing in its low-power mode. $Wakeup$ is a measurement in operation.

Let Sys be the system model of a wireless sensor node (hardware and software parts). We loop over the respective trace models for obtaining traces of significant lengths, and we verify: (a) $Idle \models$ `E<> final`, (b) $Wakeup \models$ `E<> final`, (c) $Sys||Idle \models$ `E<> final` and (d) $Sys||Wakeup \models$ `E<> final`. `final` is the label of the terminal location of the (extended) trace models.[1] Fig. 2a shows the number of reachable states in relation to the number of locations of the trace models, when verifying the above property. The figure clearly demonstrates that the number of stored states is proportional to the number of locations of the respective trace model. As shown in Fig. 2a, this problem exacerbates when there is more activity in the trace model: composite $Idle||Sys$ does not expose the non-determinism inherent in the system model, whereas composite $Wakeup||Sys$ does as seen in its steep slope.

[1] We verified these properties with Uppaal using various state space search options and the results are similar.

Sensitivity w. r. t. the Modeled Non-determinism: For investigating the effect of non-determinism we carried out an experiment with two communicating nodes. We modeled two interacting sensor nodes and checked the conformance w. r. t. the synchronously obtained series of measurements by verify $Sys^1 \| TM^1 \| Sys^2 \| TM^2 \models \texttt{E<>} (\texttt{final}^1 \vee \texttt{final}^2)$. Sys^i is the model of sensor node i (component models for hardware and software parts), TM^i is the trace model derived from the power measurement measured on sensor node i, and $\texttt{final}^i$ is the final location of trace model $TM\ i$. The result of this benchmark is shown in Fig.2b. Unicast and broadcast are concurrent measurements from two sensor nodes as described in Sec. 4. After reaching the 30'th location of TM^1, the trace model for the first sensor node, the model checker already stored $17,986,721$ states. This prohibits timed verification of more complex models as memory of commodity computers is limited. The experiments illustrate that quantitative conformance testing is severely hampered by state space explosion.

3 Segmented State Space Traversal

In the following we formally develop a segmented reachability scheme allowing us to verify the conformance of system models and measurements obtained from the implementation. With this strategy we intend to keep the memory consumption of a model checker below a threshold allowing the conformance testing of longer time series of measurements and more complex systems.

3.1 Concept and Definitions

Preliminaries: The proposed conformance test is organized as a reachability check, i. e. it answers the question if for a model $Sys||TM$ a state space traversal can generate a path $\pi(s^0\ )$ ending in a state s_t with the dedicated location $\texttt{final}$ of TM marked as active in s_t (This means that in Eq. 1 we replace model A with $Sys||TM$ and α with $\texttt{final}$). Now we refine the notation for a path as follows: $\pi(\boldsymbol{s}^0) := \boldsymbol{s}^0 \xrightarrow{\gamma} \boldsymbol{s}^i \xrightarrow{\gamma} \boldsymbol{s}^j \xrightarrow{\gamma} \boldsymbol{s}^k \ldots \xrightarrow{\gamma} \boldsymbol{s}^l_t$; where γ-transitions are either abstract delay transitions or discrete transitions referring to edge executions in the TA and i, j, k, l are some state indexes. As prefix of a path w. r. t. to a state $\boldsymbol{s}$ we define the initial path fragment up to state $\boldsymbol{s}$, not including state $\boldsymbol{s}$.

TM is a TA with a single, linear time-line of development, i. e. no alternative system evolution is possible. Let the i'th location of TM be equipped with identifier tm_i and let t be the index of the terminal location of TM. Let the edge connecting location tm_i with location tm_{i+1} be equipped with action label m_i. In the following, we refer to these edges as m-edges. For convenience we also equip location tm_t with the label $\texttt{final}$ as this dedicated location is terminal, i. e. it possesses no outgoing edge. Let us denote the single clock of TM as y. The time duration during which location tm_j $(0 \leq j < t)$, is marked active is defined by the obtained measurement series, where in TM we enforce this by a respective location invariant ($y \leq c_{tm_j}$ with $c_{tm_j} \in \mathbb{N}$ as clock constant). The location invariant, together with a corresponding edge guard ($y == c_{tm_j}$) of the outgoing edge m_j yields that TM resides exactly c_{tm_j} time units in location tm_j; y is reset upon the execution of the m-edges.

The Sweep-Line Criterion: For exemplification one may consider a state space traversal scheme, where only few system states can be stored permanently in memory. Such a scheme will not terminate as soon as there is a loop contained in $\mathcal{TS}^{Sys||TM}$ that has a diameter that exceeds the number of states which can be keept in the memory at the same time. Let K be the largest number of different states visited on a path $\pi(\boldsymbol{s}_{tm_{k-1}})$ and ending in a state $\boldsymbol{s}_{tm_k}$, with $k \in \{1, \ldots, t\}$. In the following we will show that in our setting any state space traversal scheme that stores at least K states will terminate, provided that the scheme terminates when all different states are kept in the memory. Intuitively this is because, for all paths going through a state $\boldsymbol{s}_{tm_k}$ a looping back to a state $\boldsymbol{s}_{tm_j}$ with $j < k$ is not possible, due to the linear character of TM. This yields that for the verification of $Sys||TA \models$ `E<> final`, all states that are generated a priori to the execution of an m-labelled edge do not have to be stored, i. e. they can be erased from memory.

Lemma 1. *Each path $\pi(s^0_{tm_0})$ to be generated for $Sys||TM$ and ending in a state $\boldsymbol{s}^k_{\mathtt{final}}$ has to be of the following kind:*

$$s^0_{tm_0} \xrightarrow{\gamma} s^1_{tm_0} \xrightarrow{\gamma} s^2_{tm_0} \xrightarrow{\gamma} \ldots \xrightarrow{m_1} s^a_{tm_1} \xrightarrow{\gamma} \ldots \xrightarrow{m_k} s^b_{tm_k} \xrightarrow{\gamma} \ldots \xrightarrow{m_t} s^c_{\mathtt{final}} \qquad (2)$$

where a γ-transition refers either to an abstract delay transition or to a discrete transition as induced by an edge execution in Sys. The m-transitions refer to transitions as induced by the executions of m-edges in TM. The upper indices for the states enumerate the different states seen on path $\pi(s^0)$, where in the following on the occurrence of an m-transition this index is reset to 0, i. e. $a, b, c = 0$. In the following we denote such a path as witness (cf. to Fig. 3 for an example).

Proof: The lemma above is correct as location `final` is only reachable *iff* location tm_t is reachable. This later location is only reachable *iff* location tm_{t-1} is reachable, etc.. □

A path that ends in a state where location tm_k is active is denoted in the following as tm_k-witness.

Lemma 2. *Each path $\pi(\boldsymbol{s}^0)$ constructible in $\mathcal{TS}^{Sys||TM}$ is a tm_k-witness.*

Proof: The initial state $\boldsymbol{s}^0_{tm_0}$ is a tm_k-witness, namely for $k = 0$. As all paths are terminal and ending in a deadlock with tm_k marked as active, each path must be a tm_k-witnesses. □

In this setting one may find the following property:

Theorem 1. *Discarding all states generated a priori to the generation of state $\boldsymbol{s}^0_{tm_{i+1}}$ does not interfere with the termination of the applied state space traversal scheme, provided that the scheme would have terminated when all different states of $\mathcal{TS}^{Sys||TM}$ would have been stored.*

Proof: For any tm_k witness it holds that the pairs $(m_k, s^0_{tm_k})$ appearing on such a path are unique. This is because location tm_k is marked for the very first time, i. e. there is no other state in the prefix of π w. r. t. $\boldsymbol{s}^0_{tm_k}$ resp., with tm_k marked

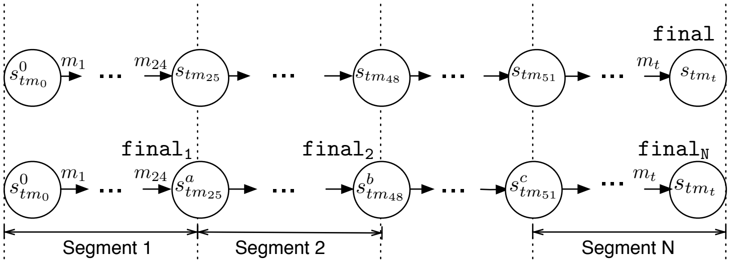


Fig. 3. Example of segmenting a trace model

active. Hence on the execution of edge m_k one cannot loop back to any state previously seen on π. As a consequence to this, all previously visited states do not have to be stored or can be erased from memory respectively, as a looping back is not possible. □

Virtual Segmentation: The above theorem features segmentation of trace models that is performed as follows: we equip n arbitrarily selected and non-adjacent locations with the dedicated label $\texttt{final}_j$, where index $j \in \mathbb{N}$ is organized in an ascending order w. r. t. the labels of the picked locations. The terminal location labeled with tm_t, or `final` respectively, is always included in this selection. For an example, please refer to Fig. 3. This figure illustrates the effect of the above labeling scheme w. r. t. to a witness that confirms $Sys||TA \models \texttt{E<>}\ \texttt{final}$.

Initial Configurations and Their Generation: Let the states $s_{\texttt{final}_j}$, i. e. states where location $final_j$ is marked active, be the start and ending points of segment-wise conformance tests. It is evident that each time one validates the conformance of a segment j of TM, which is the TA starting in location $\texttt{final}_{j-1}$ and ends in location $\texttt{final}_j$, it is necessary to restore the active locations of Sys, the valuations of the clocks, as well as the values held by the variables when reaching state $s_{\texttt{final}_{j-1}}$. In case of example Fig. 3 one needs to restore state $s^0_{\texttt{final}_1}, s^0_{\texttt{final}_2}, \ldots$ (here $s^a_{tm_{25}}, s^b_{tm_{48}}, \ldots$) when validating segment 1, 2, ... In fact, we need to restore the sets of active locations, the clock regions and possible variable values. It is interesting to note that there are sets of states of the kind $s^0_{\texttt{final}_j}$ as there might be sets of witnesses contained in Π. In the following, we generically denote the triples of active locations, clock regions and variable vectors associated with a state $s^0_{\texttt{final}_j}$ as a configuration c^i_j, where i is the index of the configuration and j the index of the segments. Label c_j denotes the set of configurations of the j'th segment.

A configuration c^i_j for initializing segment j is obtained by generating a counter-example using formula $[Sys||TM](c^i_{j-1}) \models \neg\texttt{E<>}\left(\texttt{final}_{\texttt{j-1}}\right)$.[2] The

[2] In Uppaal we need to reformulate the formula: $[Sys||TM](c^i_{j-1}) \models \texttt{A}\square\neg\,(\texttt{final}_{\texttt{j-1}})$. Note that this is straightforward due to the duality of the existential and all-quantifier in combination with the negation.

final state $s_{\mathtt{final}_{j-1}}$ of the provided counterexample contains such a configuration c^i_j. The expression $[Sys||TM](c^i_{j-1})$ denotes the initialization of composite $Sys||TM$ where configuration c^i_{j-1} is a configuration deduced in a previous run; configuration c^1_1 is defined by the initial system state.

As there are potentially many counterexamples producible when testing segment $j-1$, there are potentially many initial configurations for starting the conformance test of segment j. However, not each configuration c^i_{j-1} may actually allow us to reach state $s^0_{\mathtt{final}_j}$. In such cases, new configurations (if available) must be used as a starting point. New configurations can be determined by re-verifying the preceding segment $j-1$, but now excluding all previously used configurations stored in set c_{j-1}:

$$[Sys||TM](c^k_{j-2}) \models \neg\mathtt{E<>}\left(\mathtt{final_{j-1}} \wedge \bigvee_{\mathtt{i=1}}^{\mathtt{|c_j|}} \neg\mathtt{c^i_{j-1}}\right),$$

where $c^k_{j-2} \in c_{j-2}$ addresses some valid (initial) configuration for segment $j-2$. The negation of used configurations is necessary, as the model checker needs to be prevented from re-generating a previously used counter-example, i. e. we need to enforce the generation of a new configuration.

Digitization of Clock Regions: If one only employs clock constraints of the kind $x \bowtie c$ with $\bowtie \in \{\leq, \geq\}$, clocks can be digitized without loss of generality as far as the reachability of locations is concerned [7,2]. This allows one to sample clock values from configurations, rather than re-generating the complete clock region. This might be essential as initialization of clocks is difficult in timed model checkers and may lead to very complex models when using inequalities for clock initialization.

Exploiting a Domain-Specific Property: For low-power CPS, there is usually a dedicated recurrent operational mode, the low-power sleep mode. Here, the system does not need to perform any tasks and hence all hardware components are transferred into their low-power sleep mode. The power consumption in this low-power mode is unique and can be unambiguously associated with a measured value in the trace model, as well as with a set of locations in the network of TA modeling the CPS. In between subsequent low-power sleep modes, several operations are performed that have different impact on the system state, e. g. setting variables or resetting timers. We propose to associate a unique measurement with this unique set of locations and mark the corresponding measurement location in the trace model TM. Now virtual segmentation of the trace model TM can be facilitated using this property: we label each location of TM indicating the low-power mode with label $\mathtt{final}_j$. In case of multiple simultaneously measured traces that are tested for conformance in the same run, it is clear that segments are only created at locations of the trace models, where all trace models flag the recurrent, low-power mode. Such points in time usually exist in practice, as low-power CPS typically reside most of their deployment time ($\geq 90\%$) in the low-power mode.

Algorithm 1. Conformance check of a virtually segmented trace model

```
procedure SegmentedTraceInclusionCHK($Sys, s^0, TM, \{tm_1 \dots tm_m\}$)
    $excluded_{\{1,\dots,m\}} = \{\emptyset, \dots, \emptyset\}$
    allocate $Stack4Configurations$
    $j = 1,\ configuration :=$ ExtractConfiguration($s^0$)
    while $true$ do
        $\pi :=$ verifyta $\left([Sys||TM](configuration) \models \neg\texttt{E<>}\left(\texttt{final}_\texttt{j} \wedge \left(\bigvee_\texttt{k} \neg excluded_{\texttt{j,k}}\right)\right)\right)$
        if $\pi \neq \emptyset$ then
            if $j = m$ return $true$ end if
            push($Stack4Configurations, configuration$)
            $s_t :=$ GetLastState($\pi$)
            $configuration :=$ ExtractConfiguration($s_t$)
            $j := j + 1$
            continue
        else
            if $j = 1$ return $false$ end if
            $excluded_j := excluded_j \cup configuration$
            $configuration :=$ pop($Stack4Configurations$)
            $j = j - 1$
        end if
    end while
end procedure
```

3.2 The Procedure

A trace model is conformant w. r. t. a system model Sys *iff* all segments can be concatenated to a single witnesses and each final state of a segment coincides with the initial state of the preceding segment, i. e. formally:

$$Sys||TM \models \texttt{E<>}\,\texttt{final} \Leftrightarrow \exists\{c_1^{k_1}, \dots, c_n^{k_n}\} : \left(\bigwedge_{i:=2}^{n} [Sys||TM](c_{i-1}^{k_{i-1}}) \models \texttt{E<>}\left(\texttt{final}_\texttt{i} \wedge \texttt{c}_\texttt{i}^{\texttt{k}_\texttt{i}}\right)\right),$$

where c_1^1 $(= s^0)$ is the initial system configuration. Configuration $c_i^{k_i}$ is one specific configuration from the set of valid configurations of the segment, i. e. $c_i^{k_i} \in c_i$. Configuration $c_n^{k_n}$ is `true`. The above formula allows one to validate trace models in a sequential, segmented fashion, where a configuration is derived from a segment and carried over when resuming with the validation of the following segment. Overall this considerably reduces the peak memory consumption of the reachability problem inherent to the proposed conformance testing, as previously visited segments and their states can be ignored. With a segmented version of state space traversal, the peak memory consumption is reduced to the maximum of the peak memory consumptions of the segments.

Algorithm: We propose a depth-first search approach for validating the segments in a sequential fashion: a single configuration is selected and directly applied to the consecutive segment. Once the validation of a segment fails the algorithm has to backtrack and resumes with a preceding segment for generating a new configuration. The procedure terminates either if one successfully showed that the last segment allows for the reachability of location $\texttt{final}_n$ or the initial configuration is unable to provide a new configuration and maintaining the reachability of location $\texttt{final}_1$.

The pseudo-code shown as Algo. 1 implements this functionality as follows: the algorithm takes a system model Sys, the trace model TM, the initial state s^0 and m location labels that identify the starting point of the (virtual) trace model segments ($\{tm_1 \dots tm_m\}$) as inputs. `SegmentedTraceInclusionCHK` iteratively checks the conformance of the virtual trace segments and the provided system model. As main data structures it employs a multi-set *excluded* and a stack *Stack4Configurations*. The sets of *excluded* are used for storing those configurations that did not lead to a positive conformance test w. r. t. a specific segment. The stack *Stack4Configurations* is used for remembering the valid configurations of the preceding segments. The main functionality of the algorithm is provided by the **while** loop: at first we call a model checker, here Uppaal's `verifyta`, and check if the current segment can be validated. This is done by generating a counterexample, a path π, using the negated reachability check of location $\texttt{final}_j$. We use the function $\texttt{GetLastState}(\pi)$ to determine the last state of a path π. We use the function $\texttt{ExtractConfiguration}(s)$ to generate a configuration of a state s. Note that we initialize the models Sys and TM appropriately, i. e. with the configuration derived from state s^0. In case of a positive answer to the conformance check ($\pi \neq \emptyset$) of virtual segment j, the algorithm prepares the check of the following segment and proceeds.

If there is no witness ($\pi = \emptyset$), i. e. the current segment cannot be validated by using the current configuration, the algorithm backtracks and tries to generate a new configuration for the current segment, by excluding previously unsuccessful configurations in the query to the model checker. The negative configurations, i. e. the ones that did not produce a witness, are stored in the respective set as contained in multi-set *exclude*. The statement $\bigvee_k \neg exclude_{j,k}$ as employed in the query provided to the model checker guarantees the exclusion of previously "unsuccessful" configurations. One may note that the elements of $exclude_j$ in fact refer not only to clock constraints, but also to variable values to be excluded from the witness to be produced.

4 Case Study

In this section we investigate the applicability of the proposed segmentation for testing the conformance of synchronously measured power traces of two communicating sensor nodes, i. e. we test whether $Sys^1 \| TM^1 \| Sys^2 \| TM^2 \models \texttt{E<>}(\texttt{final}^1 \wedge \texttt{final}^2)$ holds. One may already note that the conformance of two communicating sensor nodes with corresponding simultaneously measured power consumption cannot be verified by Uppaal in a single execution, as the memory demand exceeds the capabilities of commodity computers.

Testing communicating sensor nodes: A fundamental property of sensor networks is their low-power operation. Since the radio is the major contributor to power consumption, the focus of this case study is the Medium Access (MAC) layer. MAC protocols trade off bandwidth for energy by duty-cycling the radio. Our focus is on the predominant class of random-access MAC protocols. When testing such MAC protocols, most test cases can be formulated with two or three

Table 1. Comparing power trace locations (Loc.) and states stored (States)

Model	**Naive**		**Segmented**			
	Locations	States	Locations	States	Segments	Σ States
Unicast	1213	Fails	$\leq$217	$\leq$1,881,237	15	5,828,634
Broadcast	6957	Fails	$\leq$6317	$\leq$2,614,433	8	6,318,540

sensor nodes. A sender and a receiver are fundamental and a third node may be added to test for interference, hidden terminals, etc.. This case study tests for basic functionality of the MAC protocol using two sensor nodes.

The case study uses a previously developed low-power MAC protocol [10], the so-called Harvester, that employs a low-power listening (LPL) MAC. This implies that a node may only receive at certain times, when its radio is turned on. The time between two consecutive wakeups is called the wakeup interval T_W. In particular, Harvester uses a variant of a synchronized low-power MAC protocol. In this MAC protocol scheme, nodes sleep for most of the time, yet wake up and turn on their radio after a given wakeup period to check for ongoing traffic. The MAC protocol offers two distinct operations to the sensor node software: (a) broadcast operations that are used to send a message to all nodes in the environment and (b) unicast operations, where a node sends a message to a specific neighbor. In steady-state operation, nodes only send a unicast message to another node shortly before this other node wakes up to be ready for reception. A node stops its unicasts immediately after receiving indication of reception from the addressed node. Broadcasts address all neighbors and are sent therefore for the complete wakeup period T_W in order to guarantee that all nodes in the neighborhood receive this message. Hence, a broadcast takes longer than a unicast, since it needs to last a complete wakeup period. This allows the following high-level modeling:

- We define $T_w = [0.96s, 1s]$, where nodes have to wake up each T_w.
- A sender wanting to send a unicast packet is synchronized to its receiver.
- A broadcast may start at any time and is sent for a complete T_w.
- After a broadcast the wake-up cycle of the sender may shift in time.
- After sending, receiving and listening the radio goes back to sleep.

The underlying radio and the basic MAC functionality is implemented in a radio component model, including the timing of receive, unicast and broadcast operations. The higher level functionality of periodic wakeups and re-synchronization after transmissions is modeled in a radio software model. All models including software and hardware models for the radio are adapted from [12].

Experimental setup: We monitor two communicating sensor nodes, one that only sends unicast and broadcast messages (node 10) and the other only receiving (node 12). These sensor nodes are fed by a constant voltage from a DC Power Supply (Agilent E3631A). We use two channels, one per sensor node, of a Tektronix MSO4054 Mixed Signal Oscilloscope for sampling power consumption at a rate of 50kS/s, i. e. every 20 us. The low current draw of the sensor node

Table 2. Comparison of runtimes on a 2.93GHz Intel Xeon core

Testcase	**Naive**	**Depth-Fist Search**	**Segmented**
Unicast	Out of memory	$43541s$	$4957s$
Broadcast	Out of memory	Does not finish [3]	$2932s$

combined with the noisiness of the scope in this low value ranges, necessitates an amplification of the sensor nodes' current draw. We use a Maxim MAX9922 and change the Sense resistor to 1Ω to be comparable to previous experiments [12].

Results: There are two test cases that include the wakeup behavior of two nodes: for the first test case (Unicast) there is a single unicast sent and for the second test case (Broadcast) there is a single broadcast sent. Other than these communication events, both nodes merely perform a periodic wakeup for listening. Table 1 summarizes the experiments to characterize the measured traces. First, we can see that a "naive" approach always fails due to an "Out of memory" exception (having a high exploration load of 7,520,099 and 6,942,903 states respectively) and hence is not applicable, even for simple test cases. The right part of Table 1 shows that using our segmented trace inclusion checking the traces are partitioned into 15 and 8 segments respectively. By segmenting the power trace models, the number of locations per segment is limited. In turn, the number of states per segment is limited. The maximal number of stored states is 1,881,237 states for the unicast (for 88 locations in TM^1_8 and 4 locations in TM^2_8) and $2,614,433$ for the broadcast test case (for 6265 locations in TM^1_2 and 52 locations in TM^2_2; with the lower index referring to the segment number).

Additionally, we performed runtime experiments for testing the conformance shown in Table 2, where we compare the segmented conformance test to "naive" Uppaal exploration as well as a DFS exploration. In both test cases, our tool checks the conformance without any backtracking. This is not surprising: *(i)* Each segment has a well defined initial state w. r. t. to system locations - the low-power mode; this is independent of previous behavior. *(ii)* Due to the uncertainty in measurements and variability in HW and SW, the formal model features (relaxed) bounds on timing requirements. As an example for the radio software, the invariant on the low-power state is $t \in [47500, 50000]$.

Figure 4 depicts the results for each segment, i. e. each data point represents the validation of a segment model ('x' for the Unicast and 'o' for the Broadcast test case) w. r. t. its number of locations of TM^1 and the number of stored states by the model checker. The lines in the figure repeat the results from Fig. 2 b) for comparison, where the exploration aborts after $\approx$ 40 locations are visited within the power trace model TM^1. Fig. 4 also depicts the memory wall, an experimental barrier for the naive checking of $Sys^1 \| TM^1 \| Sys^2 \| TM^2 \models$ $\texttt{E<>}(\texttt{final}^1 \wedge \texttt{final}^2)$. It is this barrier that limits the number of state visits in a single run of the model checker and that prohibits a scaling of quantitative conformance testing to more complex models. However, due to the proposed

[3] We stopped the run after 6 days, i. e. $518400s$.

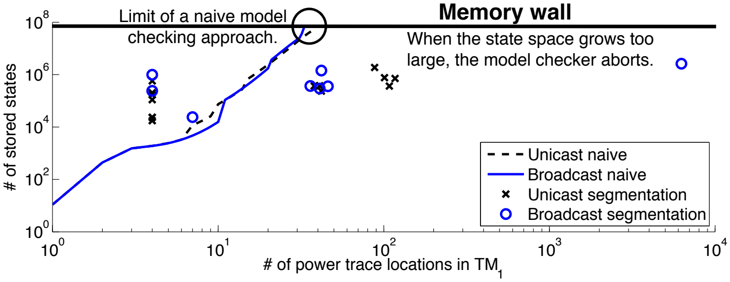


Fig. 4. Log-log plot of number of states stored by Uppaal given a number of locations

segmented trace inclusion check, the individual executions of the model checker, i. e. checking of individual segments of the TMs, stay well below this memory wall making conformance testing feasible in case of more complex models and longer measurements.

5 Summary

For conformance testing we have a linear structure of the measured trace. This paper has shown how to exploit this problem inherent structure and thereby keeping the memory requirement of conformance testing of complex system below a threshold. This works particularly well for low-power CPS, as they feature a uniquely identifiable low-power mode. In order to check the conformance of segments in isolation, the initial configuration for each segment, i. e. locations as well as clock and variable valuations has to be restored. We have shown how to determine the configurations and presented an algorithm for the sequential checking of trace segments. We validated the approach on power traces of communicating wireless sensor nodes.

Acknowledgments. We thank the anonymous reviewers for their valuable feedback that helped to improve this paper. This work was supported by NCCR-MICS (SNSF grant 5005-67322) and the Nano-Tera.ch X-Sense project.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
2. Alur, R., Madhusudan, P.: Decision problems for timed automata: A survey. In: Proc. of SFM 2004, pp. 1–24 (2004)
3. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
4. Bengtsson, J.E., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)

5. Bensalem, S., Bozga, M., Krichen, M., Tripakis, S.: Testing conformance of real-time applications by automatic generation of observers. In: Runtime Verification (RV 2004), vol. 113, pp. 23–43 (2004)
6. Christensen, S., Kristensen, L., Mailund, T.: A sweep-line method for state space exploration. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 450–464. Springer, Heidelberg (2001)
7. Henzinger, T.A., Manna, Z., Pnueli, A.: What good are digital clocks? In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 545–558. Springer, Heidelberg (1992)
8. Larsen, B.N.K.G., Mikucionis, M.: Uppaal Tron User Manual. In: CISS, BRICS, Aalborg University, Aalborg, Denmark (2009)
9. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. Formal Methods in System Design 34(3), 238–304 (2009)
10. Lim, R., Woehrle, M., Meier, A., Beutel, J.: Poster abstract: Harvester - energy savings through synchronized low-power listening. In: Adjunct Proc. EWSN 2009, pp. 29–30 (2009)
11. McMillan, K.L.: Lazy annotation for program testing and verification. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 104–118. Springer, Heidelberg (2010)
12. Woehrle, M., Lampka, K., Thiele, L.: Exploiting timed automata for conformance testing of power measurements. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 275–290. Springer, Heidelberg (2009)
13. Woehrle, M., Lampka, K., Thiele, L.: Conformance testing for cyber-physical systems. ACM Transactions on Embedded Computing Systems (2011)
14. Yovine, S.: Model checking timed automata. In: Europ. Educational Forum: School on Embedded Systems, pp. 114–152 (1998)

Event Clock Automata: From Theory to Practice*

Gilles Geeraerts[1,**], Jean-François Raskin[1], and Nathalie Sznajder[2]

[1] Université Libre Bruxelles, Département d'Informatique, Brussels, Belgium
[2] Université Pierre et Marie Curie, UMR CNRS 7606, LIP6, Paris, France
{gigeerae,jraskin}@ulb.ac.be, nathalie.sznajder@lip6.fr

Abstract. *Event clock automata* (ECA) are a model for *timed languages* that has been introduced by Alur, Fix and Henzinger as an alternative to *timed automata*, with better theoretical properties (for instance, ECA are determinizable while timed automata are not). In this paper, we *revisit* and *extend* the theory of ECA. We first prove that *no finite time abstract language equivalence* exists for ECA, thereby disproving a claim in the original work on ECA. This means in particular that regions *do not form a time abstract bisimulation*. Nevertheless, we show that regions can still be used to build a finite automaton recognizing the *untimed language of an* ECA. Then, we extend the classical notions of *zones* and *DBMs* to let them handle event clocks instead of plain clocks (as in timed automata) by introducing *event zones* and *Event DBMs* (EDBMs). We discuss algorithms to handle event zones represented as EDBMs, as well as (semi-) algorithms based on EDBMs to decide language emptiness of ECA.

1 Introduction

Timed automata have been introduced by Alur and Dill in the early nineties [2] and are a successful and popular model to reason about *timed behaviors* of computer systems. Where finite automata represent behaviors by finite sequences of actions, timed automata define sets of *timed words* (called *timed languages*) that are finite sequences of actions, each paired with a real time stamp. To this end, timed automata extend finite automata with a finite set of real valued clocks, that can be tested and reset with each action of the system. The theory of timed automata is now well developed [1]. The algorithms to analyse timed automata have been implemented in several tools such as Kronos [7] or UppAal (which is increasingly applied in industrial case studies) [4].

Timed automata, however, suffer from certain weaknesses, at least from the theoretical point of view. As a matter of fact, timed automata are *not determinizable* and *cannot be complemented* in general [2]. Intuitively, this stems from the fact that the reset of the clocks cannot be made deterministic wrt the word being read. Indeed, from a

[*] Work supported by the projects: (i) QUASIMODO (FP7- ICT-STREP-214755), Quasimodo: "Quantitative System Properties in Model-Driven-Design of Embedded", http://www.quasimodo.aau.dk/, (ii) GASICS (ESF-EUROCORES LogiCCC), Gasics: "Games for Analysis and Synthesis of Interactive Computational Systems", http://www.ulb.ac.be/di/gasics/ and (iii) Moves: "Fundamental Issues in Modelling, Verification and Evolution of Software", http://moves.ulb.ac.be, a PAI program funded by the Federal Belgian Government.

[**] Partly supported by a 'Crédit aux chercheurs' from the Belgian FRS/F.N.R.S.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 209–224, 2011.

given location, there can be two transitions, labeled by the same action a but different reset sets.

This observation has prompted Alur, Fix and Henzinger to introduce the class of *event clock automata* (ECA for short) [3], as an alternative model for timed languages. Unlike timed automata, ECA force the clock resets to be strongly linked to the occurrences of actions. More precisely, for each action a of the system, there are two clocks $\overleftarrow{x_a}$ and $\overrightarrow{x_a}$ in an ECA: $\overleftarrow{x_a}$ is the *history clock* of a and *always records the time elapsed since the last occurrence of* a. Symmetrically, $\overrightarrow{x_a}$ is the *prophecy clock* for a, and *always predicts the time distance up to the next occurrence of* a. As a consequence, while history clocks see their values *increase* with time elapsing (like clocks in timed automata do), the values of prophecy clocks *decrease over time*. However, this scheme ensures that the value of any clock is uniquely determined at any point in the timed word being read, no matter what path is being followed in the ECA. A nice consequence of this definition is that ECA are *determinizable* [3]. While the theory of ECA has witnessed some developments [14,11,16,9,12] since the seminal paper, no tool is available that exploits the full power of event clocks (the only tool we are aware of is TEMPO [15] and it is restricted to *event-recording automata*, i.e. ECA with history clocks only).

In this paper, we revisit and extend the theory of ECA, with the hope to make it more practical and amenable to implementation. A widespread belief [3] about ECA and their analysis is that ECA are similar enough to timed automata that the classical techniques (such as regions, zones or DBMs) developed for them can readily be applied to ECA. The present research, however, highlights *fundamental discrepancies* between timed automata and ECA:

1. First, we show that *there is no finite time abstract language equivalence* on the valuations of *event clocks*, whereas the region equivalence [2] *is* a finite time abstract language equivalence for timed automata. This implies, in particular, that *regions do not form a finite time-abstract bisimulation for* ECA , thereby contradicting a claim found in the original paper on ECA [3].
2. With timed automata, checking language emptiness can be done by building the so-called region automaton [2] which recognizes $\mathsf{Untime}(L(A))$, the untimed version of A's timed language. A consequence of the surprising result of point 1 is that, for some ECA A, the *region automaton recognizes a strict subset of* $\mathsf{Untime}(L(A))$. Thus, the region automaton (as defined in [2]) *is not a sound construction for checking language emptiness of* ECA . We show however that a slight modification of the original definition (that we call the *existential region automaton*) allows to recover $\mathsf{Untime}(L(A))$. Unlike the timed automata case, our proof cannot rely on bisimulation arguments, and requires original techniques.
3. Efficient algorithms to analyze timed automata are best implemented using *zones* [1], that are in turn represented by *DBMs* [10]. Unfortunately, zones and DBMs cannot be directly applied to ECA. Indeed, a zone is, roughly speaking, a conjunction of constraints of the form $x - y \prec c$, where x, y are clocks, $\prec$ is either $<$ or $\leq$ and c is an integer. This makes sense in the case of timed automata, since the difference of two clock values is an invariant with time elapsing. This is not the case when we consider event clocks, as *prophecy and history clocks evolve in opposite directions with time elapsing*. Thus, we introduce the notions of event-zones and

Event DBMs that can handle constraints of the form $x + y \prec c$, when x and y are of different types.

4. In the case of timed automata two basic, zone-based algorithms for solving language emptiness have been studied: the *forward analysis algorithm* that iteratively computes all the states reachable from the initial state, and the *backward analysis algorithm* that computes all the states that can reach an accepting state. While the former might not terminate in general, the latter is guaranteed to terminate [1]. We show that this is not the case anymore with ECA: *both algorithms might not terminate* again because of event clocks evolving in opposite directions.

These observations reflect the structure of the paper. We close it by discussing the possibility to define *widening operators*, adapted from the *closure by region*, and the *k-approximation* that have been defined for timed automata [6]. The hardest part of this future work will be to obtain a proof of correctness for these operators, since, here again, we will not be able to rely on bisimulation arguments.

Remark. Due to lack of space, most proofs have been omitted and can be found in a companion technical report [13].

2 Preliminaries

Words and timed words. An alphabet Σ is a finite set of symbols. A (finite) *word* is a finite sequence $w = w_0 w_1 \cdots w_n$ of elements of Σ. We denote the length of w by $|w|$. We denote by Σ^* the set of words over Σ. A *timed word* over Σ is a pair $\theta = (\tau, w)$ such that w is a word over Σ and $\tau = \tau_0 \tau_1 \cdots \tau_{|w|-1}$ is a word over $\mathbb{R}^{\geq 0}$ with $\tau_i \leq \tau_{i+1}$ for all $0 \leq i < |w| - 1$. We denote by $\mathsf{T}\Sigma^*$ the set of timed words over Σ. A (timed) *language* is a set of (timed) words. For a timed word $\theta = (\tau, w)$, we let $\mathsf{Untime}(\theta) = w$. For a timed language L, we let $\mathsf{Untime}(L) = \{\mathsf{Untime}(\theta) \mid \theta \in L\}$.

Event clocks. Given an alphabet Σ, we define the set of associated *event clocks* $\mathbb{C}_\Sigma = \mathbb{H}_\Sigma \cup \mathbb{P}_\Sigma$, where $\mathbb{H}_\Sigma = \{\overleftarrow{x_\sigma} \mid \sigma \in \Sigma\}$ is the set of *history clocks*, and $\mathbb{P}_\Sigma = \{\overrightarrow{x_\sigma} \mid \sigma \in \Sigma\}$ is the set of *prophecy clocks*. A *valuation* of a set of clocks is a function $v : C \to \mathbb{R}^{\geq 0} \cup \{\bot\}$, where $\bot$ means that the clock value is undefined. We denote by $\mathcal{V}(C)$ the set of all valuations of the clocks in C. For a valuation $v \in \mathcal{V}(C)$, for all $x \in \mathbb{H}_\Sigma$, we let $\langle v_1(x) \rangle = \lceil v(x) \rceil - v(x)$ and for all $x \in \mathbb{P}_\Sigma$, we let $\langle v(x) \rangle = v(x) - \lfloor v(x) \rfloor$, where $\lfloor v(x) \rfloor$ and $\lceil v(x) \rceil$ denote respectively the largest previous and smallest following integer. We also denote by $v^{\pm}$ the valuation s.t. $v^{\pm}(x) = v(x)$ for all $x \in \mathbb{H}_\Sigma$, and $v^{\pm}(x) = -v(x)$ for all $x \in \mathbb{P}_\Sigma$.

For all valuation $v \in \mathcal{V}(C)$ and all $d \in \mathbb{R}^{\geq 0}$ such that $v(x) \geq d$ for all $x \in \mathbb{P}_\Sigma \cap C$, we define the valuation $v + d$ obtained from v by letting d time units elapse: for all $x \in \mathbb{H}_\Sigma \cap C$, $(v + d)(x) = v(x) + d$ and for all $x \in \mathbb{P}_\Sigma \cap C$, $(v + d)(x) = v(x) - d$, with the convention that $\bot + d = \bot - d = \bot$. A valuation is *initial* iff $v(x) = \bot$ for all $x \in \mathbb{H}_\Sigma$, and *final* iff $v(x) = \bot$ for all $x \in \mathbb{P}_\Sigma$. We note $v[x := c]$ the valuation that matches v on all its clocks except for $v(x)$ that equals c.

An *atomic clock constraint* over $C \subseteq \mathbb{C}_\Sigma$ is either true or of the form $x \sim c$, where $x \in C$, $c \in \mathbb{N}$ and $\sim \in \{<, >, =\}$. A *clock constraint* over C is a Boolean combination of atomic clock constraints. We denote $\mathsf{Constr}(C)$ the set of all possible clock constraints over C. A valuation $v \in \mathcal{V}(C)$ satisfies a clock constraint $\psi \in \mathsf{Constr}(C)$, denoted $v \models \psi$ according to the following rules: $v \models \mathsf{true}$, $v \models x \sim c$ iff $v(x) \sim c$, $v \models \neg\psi$ iff $v \not\models \psi$, and $v \models \psi_1 \wedge \psi_2$ iff $v \models \psi_1$ and $v \models \psi_2$.

Event-clock automata. An *event-clock automaton* $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$ (ECA for short) is a tuple, where Q is a finite set of locations, $q_i \in Q$ is the initial location, Σ is an alphabet, $\delta \subseteq Q \times \Sigma \times \mathsf{Constr}(\mathbb{C}_\Sigma) \times Q$ is a finite set of edges, and $\alpha \subseteq Q$ is the set of accepting locations. We additionally require that, for each $q \in Q$, $\sigma \in \Sigma$, δ is defined for a finite number of $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$. An *extended state* (or simply state) of an ECA $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$ is a pair (q, v) where $q \in Q$ is a location, and $v \in \mathcal{V}(\mathbb{C}_\Sigma)$ is a valuation.

Runs and accepted language. The semantics of an ECA $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$ is best described by an infinite transition system $\mathsf{TS}_A = \left\langle Q^A, Q_i^A, \rightarrow, \alpha^A \right\rangle$, where $Q^A = Q \times \mathcal{V}(\mathbb{C}_\Sigma)$ is the set of extended states of A, $Q_i^A = \{(q_i, v) \mid v \text{ is initial}\}$, $\alpha^A = \{(q, v) \mid q \in \alpha \text{ and } v \text{ is final}\}$. The transition relation $\rightarrow \subseteq Q^A \times \mathbb{R}^{\geq 0} \times Q^A \cup Q^A \times \Sigma \times Q^A$ is s.t. (i) $\big((q, v), t, (q, v')\big) \in \rightarrow$ iff $v' = v + t$ (we denote this by $(q, v) \xrightarrow{t} (q, v')$), and (ii) $\big((q, v), \sigma, (q', v')\big) \in \rightarrow$ iff there is $(q, \sigma, \psi, q') \in \delta$ and $\overline{v} \in \mathcal{V}(\mathbb{C}_\Sigma)$ s.t. $\overline{v}[\overrightarrow{x_\sigma} := 0] = v$, $\overline{v}[\overleftarrow{x_\sigma} := 0] = v'$ and $\overline{v} \models \psi$ (we denote this $(q, v) \xrightarrow{\sigma} (q', v')$). We note $(q, v) \xrightarrow{t,\sigma} (q', v')$ whenever there is (q'', v'') s.t. $(q, v) \xrightarrow{t} (q'', v'') \xrightarrow{\sigma} (q', v'')$. Intuitively, this means that an history clock $\overleftarrow{x_\sigma}$ always records the time elapsed since the last occurrence of the corresponding σ event, and that a prophecy clock $\overrightarrow{x_\sigma}$ always predicts the delay up to the next occurrence of σ. Thus, when firing a σ-labeled transition, the guard must be tested against $\overline{v}$ (as defined above) because it correctly predicts the next occurrence of σ and correctly records its last occurrence (unlike v and v', as $v(\overrightarrow{x_\sigma}) = 0$ and $v'(\overleftarrow{x_\sigma}) = 0$).

A sequence $(q_0, v_0)(t_0, w_0)(q_1, v_1)(t_1, w_1)(q_2, v_2) \cdots (q_n, v_n)$ is a (q, v)-*run* of A on the timed word $\theta = (\tau, w)$ iff: $(q_0, v_0) = (q, v)$, $t_0 = \tau_0$, for any $1 \leq i \leq n - 1$: $t_i = \tau_i - \tau_{i-1}$, and for any $0 \leq i \leq n - 1$: $(q_i, v_i) \xrightarrow{t_i, w_i} (q_{i+1}, v_{i+1})$. A (q, v)-run is *initialized* iff $(q, v) \in Q_i^A$ (in this case, we simply call it a run). A (q, v)-run on θ, ending in (q_n, v_n) is *accepting* iff $(q_n, v_n) \in \alpha^A$. In this case, we say that the run accepts θ. For an ECA A and an extended state (q, v) of A, we denote by $L(A, (q, v))$ the set of timed words accepted by a (q, v)-run of A, and by $L(A)$ the set of timed words accepted by an initialized run of A.

3 Equivalence Relations for Event-Clocks

A classical technique to analyze timed transition systems is to define *time abstract equivalence relations* on the set of states, and to reason on the *quotient* transition system. In the case of *timed automata*, a fundamental concept is the *region equivalence* [2], which is a *finite time-abstract* bisimulation, and allows to decide properties of timed automata such as reachability. Contrary to a widespread belief [3], we show that the class of ECA does not benefit of these properties, as ECA **admit no finite time-abstract language equivalence**.

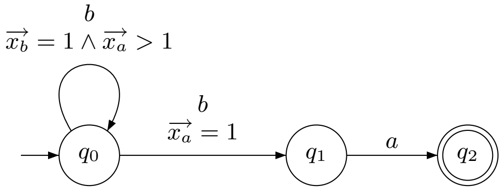

b
$\overrightarrow{x_b} = 1 \wedge \overrightarrow{x_a} > 1$
b
$\overrightarrow{x_a} = 1$
a
q_0
q_1
q_2

Fig. 1. The automaton A_{inf}

Time-abstract equivalence relations Let $\mathcal{C}$ be a class of ECA, all sharing the same alphabet Σ. We recall three equivalence notions on event clock valuations:

- $\lesssim \subseteq \mathcal{V}(\mathbb{C}_\Sigma) \times \mathcal{V}(\mathbb{C}_\Sigma)$ is a *time abstract simulation relation* for the class $\mathcal{C}$ iff, for all $\mathcal{A} \in \mathcal{C}$, for all location q of $\mathcal{A}$, for all $(v_1, v_2) \in \lesssim$, for all $t_1 \in \mathbb{R}^{\geq 0}$, for all $a \in \Sigma$: $(q, v_1) \xrightarrow{t_1, a} (q', v_1')$ implies that there exists $t_2 \in \mathbb{R}^{\geq 0}$ s.t. $(q, v_2) \xrightarrow{t_2, a} (q', v_2')$ and $v_1' \lesssim v_2'$. In this case, we say that v_2 *simulates* v_1. Finally, $\simeq \subseteq \mathcal{V}(\mathbb{C}_\Sigma) \times \mathcal{V}(\mathbb{C}_\Sigma)$ is a *time abstract simulation equivalence* iff there exists a time abstract simulation relation $\lesssim$ s.t. $\simeq = \{(v_1, v_2) \mid v_1 \lesssim v_2 \text{ and } v_2 \lesssim v_1\}$
- $\sim$ is a *time abstract bisimulation equivalence* for the class $\mathcal{C}$ iff it is a *symmetric* time abstract simulation for the class $\mathcal{C}$.
- $\approx_L \subseteq \mathcal{V}(\mathbb{C}_\Sigma) \times \mathcal{V}(\mathbb{C}_\Sigma)$ is a *time abstract language equivalence* for the class $\mathcal{C}$ iff for all $\mathcal{A} \in \mathcal{C}$, for all location q of $\mathcal{A}$, for all $(v_1, v_2) \in \approx_L$: $\mathsf{Untime}(L(q, v_1)) = \mathsf{Untime}(L(q, v_2))$

We say that an equivalence relation is *finite* iff it is of finite index. Clearly, any time abstract bisimulation is a time abstract simulation equivalence, and any time abstract simulation equivalence is a time abstract language equivalence. We prove the absence of *finite* time abstract language equivalence for ECA, thanks to A_{inf} depicted in Fig. 1:

Proposition 1. *There is no finite time abstract language equivalence for* ECA.

Proof (Sketch). Assume $\approx_L$ is a time abstract language equivalence on event-clock valuations. For any $n \in \mathbb{N}$, let v^n denote the *initial* valuation of $\mathbb{C}_{\{a,b\}}$ s.t. $v^n(\overrightarrow{x_a}) = n$ and $v^n(\overrightarrow{x_b}) = 0$, and let θ^n be the timed word $(b, 0)(b, 1)(b, 2) \cdots (b, n-1)(a, n)$. Consider the automaton A_{inf} in Fig. 1 and observe that for all $n \geq 0$, $\mathsf{Untime}(\mathsf{L}(A_{\mathsf{inf}}, (q_0, v^n))) = \mathsf{Untime}(\{\theta^n\}) = \{b^n a\}$. Hence for all the (infinitely many) pairs (i, j) with $i \neq j$: $v^i \not\approx_L v^j$, and thus $\approx_L$ is not finite. □

Corollary 2. *There is no* finite *time abstract language equivalence, no* finite *time abstract simulation equivalence and no* finite *time abstract bisimulation for* ECA.

4 Regions and Event Clocks

For the class of timed automata, the *region equivalence* has been shown to be a *finite time-abstract bisimulation*, which is used to build the so-called *region automaton*, a finite-state automaton recognizing $\mathsf{Untime}(L(A))$ for all timed automata A [2]. Corollary 2 tells us that regions are not a time-abstract bisimulation for ECA (contrary to what was claimed in [3]). Let us show that we can nevertheless rely on the notion of region to build a finite automaton recognizing $\mathsf{Untime}(L(A))$ for all ECA A.

Regions. Let us fix a set of clocks $C \subseteq \mathbb{C}_\Sigma$ and a constant $cmax \in \mathbb{N}$. We first recall two region equivalences from the literature. The former, denoted $\approx_{cmax}$, is the classical Alur-Dill region equivalence for timed automata [2] while the latter (denoted $\approx^{\angle}_{cmax}$) is adapted from Bouyer [6] and refines the former:

- For any $v_1, v_2 \in \mathcal{V}(C)$: $v_1 \approx_{cmax} v_2$ iff:
  (C1) for all $x \in C$, $v_1(x) = \bot$ iff $v_2(x) = \bot$,
  (C2) for all $x \in C$: either $v_1(x) > cmax$ and $v_2(x) > cmax$, or $\lceil v_1(x) \rceil = \lceil v_2(x) \rceil$ and $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$,
  (C3) for all x_1, $x_2 \in C$ s.t. $v_1(x_1) \leq cmax$ and $v_1(x_2) \leq cmax$: $\langle v_1(x_1) \rangle \leq \langle v_1(x_2) \rangle$ if and only if $\langle v_2(x_1) \rangle \leq \langle v_2(x_2) \rangle$.
- For all $v_1, v_2 \in \mathcal{V}(C)$: $v_1 \approx^{\angle}_{cmax} v_2$ iff: $v_1 \approx_{cmax} v_2$ and:
  (C4) For all $x_1, x_2 \in C$ s.t. $v_1(x_1) > cmax$ or $v_1(x_2) > cmax$: either we have $\left| v_1^{\pm}(x_1) - v_1^{\pm}(x_2) \right| > 2 \cdot cmax$ and $\left| v_2^{\pm}(x_1) - v_2^{\pm}(x_2) \right| > 2 \cdot cmax$; or we have $\lfloor v_1^{\pm}(x_1) - v_1^{\pm}(x_2) \rfloor = \lfloor v_2^{\pm}(x_1) - v_2^{\pm}(x_2) \rfloor$ and $\lceil v_1^{\pm}(x_1) - v_1^{\pm}(x_2) \rceil = \lceil v_2^{\pm}(x_1) - v_2^{\pm}(x_2) \rceil$.

Equivalence classes of both $\approx_{cmax}$ and $\approx^{\angle}_{cmax}$ are called *regions*. We denote by Reg $(C, cmax)$ and Reg$^{\angle}$ $(C, cmax)$ the set of regions of $\approx_{cmax}$ and $\approx^{\angle}_{cmax}$ respectively. Fig. 2 (a), (b) and (c) illustrate these two notions. Comparing (a) and (b) clearly shows how $\approx^{\angle}_{cmax}$ refines $\approx_{cmax}$ by introducing diagonal constraints between clocks larger than $cmax$. Moreover, (c) shows why we need to rely on $v_1^{\pm}$ and $v_2^{\pm}$ in C4: in this case, C contains an history and a prophecy clock that evolve in opposite directions with time elapsing. Thus, their *sum* remains constant over time (hence the $2 \cdot cmax$ in C4).

Observe that, for any $cmax$, and for any finite set of clocks C, Reg $(C, cmax)$ and Reg$^{\angle}$ $(C, cmax)$ are *finite* sets. A region r on set of clocks C is *initial* (resp. *final*) iff it contains only initial (final) valuations.

Regions are not a language equivalence. Since both notions of regions defined above are finite, Corollary 2 implies that they cannot form a language equivalence for ECA. Let us explain intuitively why it is not the case. Consider Reg $\left(\mathbb{P}_{\{a,b\}}, 1\right)$ and the two valuations v_1 and v_2 in Fig. 2 (a). Clearly, v_1 can reach the region where $\overrightarrow{x_a} = 1$ and $\overrightarrow{x_b} > 1$, while v_2 cannot. Conversely, v_2 can reach $\overrightarrow{x_a} > 1$ and $\overrightarrow{x_b} = 1$ but v_2 cannot. It is easy to build an ECA with $cmax = 1$ that distinguishes between those two cases and accepts different words. Then, consider Reg$^{\angle}$ $\left(\mathbb{P}_{\{a,b\}}, 1\right)$ and the valuations v^3 and v^4 (not shown in the figure) s.t. $v^3(\overrightarrow{x_b}) = v^4(\overrightarrow{x_b}) = 1$, $v^3(\overrightarrow{x_a}) = 4$ and $v^4(\overrightarrow{x_a}) = 5$. It is easy to see that for A_{inf} in Fig. 1: $\mathsf{Untime}(L(A_{\inf}, (q_0, v^3))) = \{bbba\} \neq \{bbbba\} = \mathsf{Untime}(L(A_{\mathsf{inf}}, (q_0, v^4)))$, although v^3 and v^4 belong to the same region. Indeed, from v^3, the (q_0, q_0) loop can be fired 3 times before we reach $\overrightarrow{x_a} = 1$ and the (q_0, q_1) edge can be fired. However, the (q_0, q_0) loop has to be fired 4 times from v^4 before we reach $\overrightarrow{x_a} = 1$ and the (q_0, q_1) edge can be fired. Remark that these are essentially the same arguments as in the proof of Proposition 1. These two examples illustrate the issue with *prophecy clocks* and regions. Roughly speaking, to keep the set of regions finite, valuations where the clocks are *too large* (for instance, $> cmax$ in the case of Reg $(C, cmax)$) belong to the same region. This is not a problem for history clocks as an history clock larger than $cmax$ remains over $cmax$ with time elapsing. This is not the case for prophecy clocks whose values *decrease with time elapsing*: eventually,

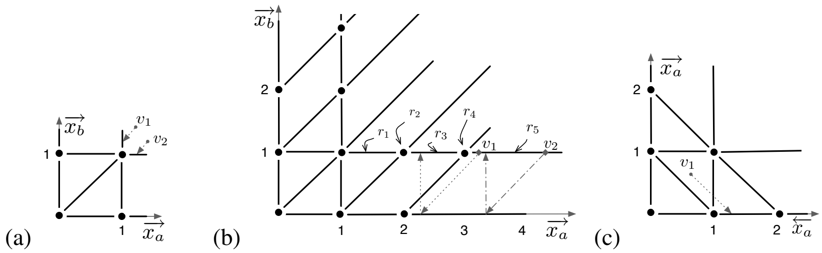


Fig. 2. The sets of regions (a) $\mathsf{Reg}\left(\mathbb{P}_{\{a,b\}}, 1\right)$, (b) $\mathsf{Reg}^{\angle}\left(\mathbb{P}_{\{a,b\}}, 1\right)$ and (c) $\mathsf{Reg}^{\angle}\left(\mathbb{C}_{\{a\}}, 1\right)$. Dotted arrows show the trajectories followed by the valuations with time elapsing. Curved arrows are used to refer to selected regions.

those clocks reach a value $\leq cmax$, but the region equivalence is too coarse to allow to predict the region they reach.

Region automata. Let us now consider the consequence of Corollary 2 on the notion of region automaton. We first define two variants of the region automaton:

Definition 3. *Let $A = \langle Q, q_i, \Sigma, \delta, \alpha\rangle$ and $\mathcal{R}$ be a set of regions on $\mathcal{V}(\mathbb{C}_\Sigma)$. Then, the **existential** (resp. **universal**) $\mathcal{R}$-*region automaton of *$\mathcal{A}$ is the finite transition system $RA(\exists, \mathcal{R}, A)$ (resp. $RA(\forall, \mathcal{R}, A)$) defined by $\left\langle Q^R, Q_i^R, \Sigma, \delta^R, \alpha^R\right\rangle$ s.t.:*

1. *$Q^R = Q \times \mathcal{R}$*
2. *$Q_i^R = \{(q_i, r) \mid r$ is an initial region$\}$*
3. *$\delta^R \subseteq Q^R \times \Sigma \times Q^R$ is s.t. $\big((q_1, r_1), a, (q_2, r_2)\big) \in \delta$ iff **there exists a valuation** (resp. **for all valuations**) $v_1 \in r_1$, there exists a time delay $t \in \mathbb{R}^{\geq 0}$ and a valuation $v_2 \in r_2$ s.t. $(q_1, v_1) \xrightarrow{t,a} (q_2, v_2)$.*
4. *$\alpha^R = \{(q, r) \mid q \in \alpha$ and r is a final region$\}$*

Let $R = \left\langle Q^R, Q_i^R, \Sigma, \delta^R, \alpha^R\right\rangle$ be a region automaton and w be an (untimed) word over Σ. A *run* of R on $w = w_0 w_1 \dots w_n$ is a finite sequence $(q_0, r_0)(q_1, r_1) \dots (q_{n+1}, r_{n+1})$ of states of R s.t.: $(q_0, r_0) \in Q_i^R$ and for all $0 \leq i \leq n$: $\big((q_i, r_i), w_i, (q_{i+1}, r_{i+1})\big) \in \delta^R$. Such a run is *accepting* iff $(q_{n+1}, r_{n+1}) \in \alpha^R$ (in that case, we say that w is accepted by R). The language $L(R)$ of R is the set of all untimed words accepted by R.

Let A be an ECA with alphabet Σ and maximal constant $cmax$. If we adapt and apply the notion of region automaton, as defined for timed automata [2], to A we obtain $RA(\forall, \mathsf{Reg}(\mathbb{C}_\Sigma, cmax), A)$. To alleviate notations, we denote it by $\mathsf{RegAut}_\forall(A)$. In the rest of the paper, we also consider three other variants: (i) $\mathsf{RegAut}_\forall^{\angle}(A) = RA(\forall, \mathsf{Reg}^{\angle}(\mathbb{C}_\Sigma, cmax), A)$, (ii) $\mathsf{RegAut}_\exists(A) = RA(\exists, \mathsf{Reg}(\mathbb{C}_\Sigma, cmax), A)$ and (iii) $\mathsf{RegAut}_\exists^{\angle}(A) = RA(\exists, \mathsf{Reg}^{\angle}(\mathbb{C}_\Sigma, cmax), A)$. Observe that, for timed automata, all these automata coincide, and thus accept the untimed language (this can be proved by a bisimulation argument) [2]. Let us see how these results adapt (or not) to ECA.

Recognized language of universal region automata. Let us show that, in general *universal* region automata *do not recognize the untimed language of the* ECA.

Lemma 4. *There is an* ECA A *such that* $L(\mathsf{RegAut}_{\forall}(A)) \subsetneq \mathsf{Untime}(L(A))$ *and such that* $L(\mathsf{RegAut}^{\angle}_{\forall}(A)) \subsetneq \mathsf{Untime}(L(A))$.

Proof (Sketch). The ECA A_{inf} in Fig. 1 enjoys this property. We detail the arguments for the second case. Since $cmax = 1$, the set of regions we consider is depicted in Fig. 2 (b) (for the valuations where clocks are $\neq \bot$). Assume there is, in $\mathsf{RegAut}^{\angle}_{\forall}(A_{\mathsf{inf}})$, an edge of the form $\big((q_0, r), b, (q_0, r')\big)$ were r is initial. This implies that $r' \in \{r_1, \dots, r_5\}$ (we refer to the names in Fig. 2), because of the guard of the (q_0, q_0) loop. Since $\mathsf{Untime}(L(A_{\mathsf{inf}})) = \{b^n a \mid n \geq 1\}$, it must be possible to accept an arbitrary number of b's from one of the (q_0, r'). Let us show that it is not the case. From r_3 and r_4 we have edges $\big((q_0, r_3), b, (q_0, r_1)\big)$ and $\big(q_0, r_4), b, (q_0, r_2)\big)$. However, there is no valuation $v \in r_1 \cup r_2$ s.t. $(v+t)(\overrightarrow{x_b}) = 0$ and $(v+t)(\overrightarrow{x_a}) > 1$ for some t. Thus, there is, in $\mathsf{RegAut}^{\angle}_{\forall}(A_{\mathsf{inf}})$, no edge of the form $\big((q_0, r), b, (q_0, r')\big)$ when $r \in \{r_1, r_2\}$. Finally, there is no edge of the form $\big((q_0, r_5), b, (q_0, r)\big)$ because *some valuations* of r_5 (such as v_1) will reach r_3 and *some others* (such as v_2) will stay in r_5 after the firing of the loop. Since we consider an *universal* automaton, (q_0, r_5) has no successor. □

Recognized language of existential region automata. Fortunately, the definition of *existential region automaton* allows us to recover a finite transition system recognizing exactly $\mathsf{Untime}(L(A))$, for all ECA A. Remark that, to establish this result, we cannot rely on bisimulation arguments. Let us show that $\mathsf{Untime}(L(A)) \subseteq L(\mathsf{RegAut}^{\angle}_{\exists}(A)) \subseteq L(\mathsf{RegAut}_{\exists}(A)) \subseteq \mathsf{Untime}(L(A))$.

The two leftmost inequalities are easily established. Let $(q_0, v_0)(t_0, w_0)(q_1, v_1)(t_1, w_1) \cdots (q_n, v_n)$ be an accepting run of A on $\theta = (\tau, w)$. Thus, $\theta \in L(A)$. For all $0 \leq i \leq n$ let r_i be the (unique) region containing v_i. Then, by definition of $\mathsf{RegAut}^{\angle}_{\exists}(A)$, $(q_0, r_0)w_0(q_1, r_1)w_1 \cdots (q_n, r_n)$ is an accepting run of $\mathsf{RegAut}^{\angle}_{\exists}(A)$ on $w = \mathsf{Untime}(\theta)$. Hence $\mathsf{Untime}(L(A)) \subseteq L(\mathsf{RegAut}^{\angle}_{\exists}(A))$. Second, since $\approx^{\angle}_{cmax}$ refines $\approx_{cmax}$, each accepting run $(q_0, r_0)w_0(q_1, r_1)w_1 \cdots (q_n, r_n)$ in $\mathsf{RegAut}^{\angle}_{\exists}(A)$ corresponds to an accepting run $(q_0, r'_0)w_0(q_1, r'_1)w_1 \cdots (q_n, r'_n)$ in $\mathsf{RegAut}_{\exists}(A)$, where for any $0 \leq i \leq n$, r'_i is the (unique) region of $\mathsf{Reg}(\mathbb{C}_{\Sigma}, cmax)$ that contains r_i. Hence, $L(\mathsf{RegAut}^{\angle}_{\exists}(A)) \subseteq L(\mathsf{RegAut}_{\exists}(A))$.

To establish $L(\mathsf{RegAut}_{\exists}(A)) \subseteq \mathsf{Untime}(L(A))$ we need to rely on the notion of *weak time successor*. The set of *weak time successors* of v by t time units is:

$$v +_{\mathrm{w}} t = \left\{ v' \;\middle|\; \forall x : \begin{array}{c} \big(x \in \mathbb{P}_{\Sigma} \text{ and } v(x) > cmax\big) \text{ implies } v'(x) > cmax - t \\ \text{and} \\ \big(x \notin \mathbb{P}_{\Sigma} \text{ or } v(x) \leq cmax \text{ or } v(x) = \bot\big) \text{ implies } v'(x) = (v+t)(x) \end{array} \right\}$$

As can be seen, weak time successors introduce non-determinism on prophecy clocks that are larger than $cmax$. So, $v +_{\mathrm{w}} t$ is a *set* of valuations. Let q be a location of an ECA. We write $(q, v) \xrightarrow{t}_{w} (q, v')$ whenever $v' \in (v +_{\mathrm{w}} t)$. Then, a sequence $(q_0, v_0)(t_0, w_0)(q_1, v_1)(t_1, w_1)(q_2, v_2) \cdots (q_n, v_n)$ is an initialized *weak run*, on $\theta = (\tau, w)$, of an ECA $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$ iff $q_0 = q_i$, v_0 is initial, $t_0 = \tau_0$, for any $1 \leq i \leq n-1$: $t_i = \tau_i - \tau_{i-1}$, and for any $0 \leq i \leq n-1$: there is (q'_i, v'_i) s.t. $(q_i, v_i) \xrightarrow{t_i}_{w} (q'_i, v'_i) \xrightarrow{w_i} (q_{i+1}, v_{i+1})$. A weak run is accepting iff $q_n \in \alpha$ and v_n is final. The weak language $\mathsf{wL}(A)$ of A is the set of all timed words θ s.t. there is an accepting weak run on θ.

Clearly, $L(A) \subseteq \mathsf{wL}(A)$ as every run is also a weak run. However, the converse also holds, since the non-determinism appears only on clocks larger than $cmax$, which the automaton cannot distinguish:

Proposition 5. *For any* ECA A*:* $L(A) = \mathsf{wL}(A)$.

Then, we prove that *weak time successors* enjoy a property which is reminiscent of time abstract bisimulation. This allows to establish Theorem 7.

Lemma 6. *Let C be a set of clocks and let $cmax$ be a natural constant. For any $v_1, v_2 \in \mathcal{V}(C)$ s.t. $v_1 \approx_{cmax} v_2$, for any $t_1 \in \mathbb{R}^{\geq 0}$, there exist t_2 and $v' \in (v_2 +_{\mathrm{w}} t_2)$ s.t. $v_1 + t_1 \approx_{cmax} v'$.*

Theorem 7. *For any* ECA $A = (\Sigma, Q, q_i, \delta, \alpha)$*:* $\mathsf{L}(\mathsf{RegAut}_{\exists}(A)) \subseteq \mathsf{Untime}(\mathsf{L}(A))$.

Proof (Sketch). For every run on w in the region automaton, we build a sequence of time stamps τ s.t. $\theta = (\tau, w)$ is in $L(A)$. The main difficulty stems from the fact that we consider an *existential automaton*: assume there are $\big((q_1, r_1), a, (q_2, r_2)\big)$ and $\big((q_2, r_2), b, (q_3, r_3)\big)$ in δ^R. Then, there are $v_1 \in r_1$, $v_1', v_2 \in r_2$ and $v_2' \in r_3$ s.t. $(q_1, v_1) \xrightarrow{a,t_1} (q_2, v_1')$ and $(q_2, v_2) \xrightarrow{b,t_2} (q_3, v_2')$ for some t_1, t_2, but *possibly with* $v_1' \neq v_2$. Building θ by induction is thus more involved than with a universal automaton. However, for any run of $\mathsf{RegAut}_{\exists}(A)$ over a word $w \in \Sigma^*$, we can inductively build, by using Lemma 6, a time sequence τ and a *weak run* of A over (τ, w) that visits the same locations. By Proposition 5, this concludes the proof. □

Size of the existential region automaton. The number of Alur-Dill regions on n clocks and with maximal constant $cmax$ is at most $R(n, cmax) = n! \times 2^n \times (2 \times cmax + 2)^n$ [2]. Adapting this result to take into account the $\perp$ value, we have: $|\mathsf{Reg}\,(\mathbb{C}_{\Sigma}, cmax)| \leq R(2 \times |\Sigma|, cmax + 1)$. Hence, the number of locations of $\mathsf{RegAut}_{\exists}(A)$ for an ECA A with m locations and alphabet Σ is at most $m \times R(2 \times |\Sigma|, cmax + 1)$. In [3], a technique is given to obtain a finite automaton recognizing $\mathsf{Untime}(L(A))$ for all ECA A: first transform A into a non-deterministic timed automaton [2] A' s.t. $L(A') = L(A)$, then compute the region automaton of A'. However, building A' incurs a blow up in the number of clocks and locations, and the size of the region automaton of A' is at most $m \times 2^K \times R(K, cmax)$ where $K = 6 \times |\Sigma| \times (cmax + 2)$ is an upper bound on the number of atomic clock constraints in A. Our construction thus yields a smaller automaton.

5 Zones and Event-Clocks

In the setting of timed automata, the *zone datastructure* [10] has been introduced as an effective way to improve the running time and memory consumption of on-the-fly algorithms for checking emptiness. In this section, we *adapt* this notion to the framework of ECA, and discuss forward and backward analysis algorithms. Roughly speaking, a *zone* is a symbolic representation for a set of clock valuations that are defined by constraints of the form $x - y \prec c$, where x, y are clocks, $\prec$ is either $<$ or $\leq$, and c is an integer constant. Keeping the difference between clock values makes sense in the setting of timed automata as all the clocks have always real values and the difference between two clock values is an invariant over the elapsing of time. To adapt the notion of zone to ECA, we need to overcome two difficulties. First, prophecy and history clocks evolve in different directions with time elapsing. Hence, it is not always the case that if $v(x) - v(y) = c$ then $(v+t)(x) - (v+t)(y) = c$ for all t (for instance if x is a prophecy clocks and y an history clock). However, the *sum* of clocks of different types is now an invariant, so event clock zones must be definable, either by constraints of the form $x - y \prec c$, if x and y are both history or both prophecy clocks, or by constraints of the form $x + y \prec c$ otherwise. Second, clocks can now take the special value $\perp$. Formally, we introduce the notion of event-zone as follows.

Definition 8. *For a set C of clocks over an alphabet Σ, an* event-zone *is a subset of $\mathcal{V}(C)$ that is defined by a conjunction of constraints of the form $x = \perp$; $x \sim c$; $x_1 - x_2 \sim c$ if $x_1, x_2 \in \mathbb{H}_\Sigma$ or $x_1, x_2 \in \mathbb{P}_\Sigma$; and $x_1 + x_2 \sim c$ if either $x_1 \in \mathbb{H}_\Sigma$ and $x_2 \in \mathbb{P}_\Sigma$ or $x_1 \in \mathbb{P}_\Sigma$ and $x_2 \in \mathbb{H}_\Sigma$, with $x, x_1, x_2 \in C$, $\sim \in \{\leq, \geq, <, >\}$ and $c \in \mathbb{Z}$.*

Event-clock Difference Bound Matrices In the context of timed automata, Difference Bound Matrices (DBMs for short) have been introduced to represent and manipulate zones [5,10]. Let us now adapt DBMs to event clocks.

Formally, an EDBM M of the set of clocks $C = \{x_1, \ldots, x_n\}$ is a $(n+1)$ square matrix of elements from $\big(\mathbb{Z} \times \{<, \leq\}\big) \cup \{(\infty, <), (\perp, =), (?, =)\}$ s.t. for all $0 \leq i, j, \leq n$: $m_{i,j} = (\perp, =)$ implies $i = 0$ or $j = 0$ (i.e., $\perp$ can only appear in the first position of a row or column). Thus, a constraint of the form $x_i = \perp$ will be encoded with either $m_{i,0} = (\perp, =)$ or $m_{0,i} = (\perp, =)$. As in the case of DBMs, we assume that the extra clock x_0 is always equal to zero. Moreover, since prophecy clocks decrease with time evolving, they are encoded by their *opposite value* in the matrix. Hence the EDBM naturally encodes *sums* of variables when the two clocks are of different types. Each element $(m_{ij}, \prec_{ij})$ of the matrix thus represents either the constraint $x_i - x_j \prec_{ij} m_{ij}$ or the constraint $x_i + x_j \prec_{ij} m_{ij}$, depending on the type of x_i and x_j. Finally, the special symbol ? encodes the fact that the variable is not constrained (it can take any real value, or the $\perp$ value). Formally, an EDBM M on set of clocks $C = \{x_1, \ldots, x_n\}$ represents the zone $[\![M]\!]$ on set of clocks C s.t. $v \in [\![M]\!]$ iff for all $0 \leq i, j \leq n$: **if** $M_{i,j} = (c, \prec)$ with $c \neq ?$ **then** $v^{\pm}(x_i) - v^{\pm}(x_j) \prec c$ (assuming $v^{\pm}(x_0)$ denotes the value 0 and assuming that for all $k \in \mathbb{Z} \cup \{\perp\}$: $\perp + k = \perp - k = k + \perp = k - \perp = \perp$). When $[\![M]\!] = \varnothing$, we say that M is *empty*. In the sequel, we also rely on the $\leq$ ordering on EDBM elements. We let $(m; \prec) \leq (m'; \prec')$ iff one of the following holds: either (i) $m' = ?$; or (ii) $m, m' \in \mathbb{Z} \cup \{\infty\}$ and $m < m'$; or (iii) $m = m'$ and either $\prec = \prec'$ or $\prec' = \leq$.

As an example, consider the two following EDBMs that both represent $x_1 = \perp \wedge 0 < x_3 - x_4 < 1 \wedge x_2 + x_4 \leq 2$ (where x_1, x_2 are prophecy clocks, and x_3, x_4 are history clocks):

$$\begin{pmatrix} (0,\leq) & (\perp,=) & (?,=) & (?,=) & (?,=) \\ (\perp,=) & (?,=) & (?,=) & (?,=) & (?,=) \\ (0,\leq) & (?,=) & (0,\leq) & (?,=) & (?,=) \\ (?,=) & (?,=) & (?,=) & (0,\leq) & (1,<) \\ (?,=) & (?,=) & (2,\leq) & (0,<) & (0,\leq) \end{pmatrix} \begin{pmatrix} (0,\leq) & (\perp,=) & (\infty,<) & (0,\leq) & (0,\leq) \\ (\perp,=) & (?,=) & (?,=) & (?,=) & (?,=) \\ (0,\leq) & (?,=) & (0,\leq) & (0,\leq) & (0,\leq) \\ (\infty,<) & (?,=) & (\infty,<) & (0,\leq) & (1,<) \\ (\infty,<) & (?,=) & (2,\leq) & (0,<) & (0,\leq) \end{pmatrix}$$

Normal form EDBMs. As in the case of DBMs, we define a *normal form* for EDBM, and show how to turn any EDBM M into a normal form EDBM M' s.t. $[\![M]\!] = [\![M']\!]$. A non-empty EDBM M is in *normal form* iff the following holds: (i) for all $1 \leq i \leq n$: $M_{i,0} = (\bot, =)$ iff $M_{0,i} = (\bot, =)$ and $M_{i,0} = (?, =)$ iff $M_{0,i} = (?, =)$, (ii) for all $1 \leq i \leq n$: $M_{i,0} \in \{(\bot, =), (?, =)\}$ implies $M_{i,j} = M_{j,i} = (?, =)$ for all $1 \leq j \leq n$, (iii) for all $1 \leq i, j \leq n$: $M_{i,j} = (?, =)$ iff either $M_{i,0} \in \{(?, =), (\bot, =)\}$ or $M_{j,0} \in \{(?, =), (\bot, =)\}$ and (iv) the matrix M' is a *normal form DBM* [10], where M' is obtained by projecting away all lines $1 \leq i \leq n$ s.t. $M_{i,0} \in \{(?, =), (\bot, =)\}$ and all columns $1 \leq j \leq n$ s.t. $M_{0,j} \in \{(?, =), (\bot, =)\}$ from M. To canonically represent the empty zone, we select a particular EDBM $M_\varnothing$ s.t. $[\![M_\varnothing]\!] = \varnothing$. For example, the latter EDBM of the above example is in normal form.

Then, given an EDBM M, Algorithm 1 allows to compute a normal form EDBM M' s.t. $[\![M]\!] = [\![M']\!]$. This algorithm relies on the function `DBMNormalise(`M,S`)`, where M is an $(\ell+1) \times (\ell+1)$ EDBM, and $S \subseteq \{0, \ldots, \ell\}$. `DBMNormalise(`$M$,$S$`)` applies the classical normalisation algorithm for DBMs [10] on the DBM obtained by projecting away from M all the lines and columns $i \notin S$. Algorithm 1 proceeds in three steps. In the first loop, we look for lines (resp. columns) i s.t. $M_{i,0}$ (resp. $M_{0,i}$) is $(\bot, =)$, meaning that there is a constraint imposing that $x_i = \bot$. In this case, the corresponding $M_{0,i}$ (resp. $M_{i,0}$) must be equal to $(\bot, =)$ too, and all the other elements in the ith line and column must contain $(?, =)$. If we find a j s.t. $M_{i,j} \neq (?, =)$ or $M_{j,i} \neq (?, =)$, then the zone is empty, and we return $M_\varnothing$. Then, in the second loop, the algorithm looks for lines (resp. columns) i with the first element equal to $(?, =)$ but containing a constraint of the form $(c, \prec)$, which imposes that the variable i must be different from $\bot$. We record this information by replacing the $(?, =)$ in $M_{i,0}$ (resp. $M_{0,i}$) by the weakest possible constraint that forces x_i to have a value different from $\bot$. This is either $(0, \leq)$ or $(\infty, <)$, depending on the type of x_i and is taken care by the `SetCst()` function. At this point the set S contains the indices of all variables that are constrained to be real. The algorithm finishes by calling the normalisation function for DBMs. Remark, in particular, that the algorithm returns $M_\varnothing$ iff M is empty which also provides us with a test for EDBM emptiness.

Proposition 9. *For all EDBM M,* `EDBMNormalise(`M`)` *returns a normal form EDBM M' s.t.* $[\![M']\!] = [\![M]\!]$.

Operations on zones. The four basic operations we need to perform on event-zones are: (i) *future* of an event-zone Z : $\overrightarrow{Z} = \{v \in \mathcal{V}(\mathbb{C}_\Sigma) \mid \exists v' \in Z, t \in \mathbb{R}^{\geq 0} : v = v' + t\}$; (ii) *past* of an event-zone Z : $\overleftarrow{Z} = \{v \in \mathcal{V}(\mathbb{C}_\Sigma) \mid \exists t \in \mathbb{R}^{\geq 0} : v + t \in Z\}$; (iii) *intersection* of two event-zones Z and Z'; and (iv) *release* of a clock x in Z: $\mathsf{rel}_x(Z) = \{v[x := d] \mid v \in Z, d \in \mathbb{R}^{\geq 0} \cup \{\bot\}\}$. Moreover, we also need to be able to test for inclusion of two zones encoded as EDBMs. Let M, M_1 and M_2 be EDBMs in normal form, on n clocks. Then:

Future. If $M = M_\varnothing$, we let $\overrightarrow{M} = M_\varnothing$. Otherwise, we let $\overrightarrow{M}$ be s.t.:

$$\overrightarrow{M}_{i,j} = \begin{cases} (0, \leq) & \text{if } M_{ij} \notin \{(\bot, =), (?, =)\}, j = 0 \text{ and } x_i \in \mathbb{P}_\Sigma \\ (\infty, <) & \text{if } M_{ij} \notin \{(\bot, =), (?, =)\}, j = 0 \text{ and } x_i \in \mathbb{H}_\Sigma \\ M_{i,j} & \text{otherwise} \end{cases}$$

```
EDBMNormalise($M$) begin
    Let $S = \{0\}$ ;
    foreach $1 \leq i \leq n$ s.t. $M_{i,0} = (\bot, =)$ or $M_{0,i} = (\bot, =)$ do
        if $\exists 1 \leq j \leq n$ s.t. $M_{i,j} \neq (?, =)$ or $M_{j,i} \neq (?, =)$ then return $M_\varnothing$;
        $M_{i,0} \leftarrow (\bot, =)$ ; $M_{0,i} \leftarrow (\bot, =)$ ;
    foreach $0 \leq i, j \leq n$ s.t. $M_{i,j} \notin \{(?, =), (\bot, =)\}$ do
        $S \leftarrow S \cup \{i, j\}$ ;
    foreach $i, j \in S$ do SetCst($M_{i,j}$) ;
    $M' \leftarrow$ DBMNormalise($M$,$S$) ;
    if $M' = Empty$ then return $M_\varnothing$ ;
    return $M'$ ;

SetCst($M_{i,j}$) begin
    if $M_{i,j} = (?, =)$ then
        if $x_i \in \mathbb{P}_\Sigma$ and ($x_j \in \mathbb{H}_\Sigma$ or $x_j = x_0$) then $M_{i,j} \leftarrow (0, \leq)$ ;
        else $M_{i,j} \leftarrow (\infty, <)$ ;
```

Algorithm 1. A normalisation algorithm for EDBMs

Past. If $M = M_\varnothing$, we let $\overleftarrow{M} = M_\varnothing$. Otherwise, we let $\overleftarrow{M}$ be s.t. for all i, j:

$$\overleftarrow{M}_{i,j} = \begin{cases} (\infty, <) & \text{if } M_{ij} \notin \{(\bot, =), (?, =)\}, i = 0 \text{ and } x_j \in \mathbb{P}_\Sigma \\ (0, \leq) & \text{if } M_{ij} \notin \{(\bot, =), (?, =)\}, i = 0 \text{ and } x_j \in \mathbb{H}_\Sigma \\ M_{i,j} & \text{otherwise} \end{cases}$$

Intersection. We consider several cases. If $M^1 = M_\varnothing$ or $M^2 = M_\varnothing$, we let $M^1 \cap M^2 = M_\varnothing$. If there are $0 \leq i, j \leq n$ s.t. $M^1_{i,j} \not\leq M^2_{i,j}$ and $M^2_{i,j} \not\leq M^1_{i,j}$, we let $M^1 \cap M^2 = M_\varnothing$ too. Otherwise, we let $M^1 \cap M^2$ be the EDBM M' s.t for all i, j: $M'_{i,j} = min(M^1_{i,j}, M^2_{i,j})$.

Release. Let x be an event clock. In the case where $M = M_\varnothing$, we let $\mathsf{rel}_x(M) = M_\varnothing$. Otherwise, we let $\mathsf{rel}_x(M)$ be the EDBM s.t. for all i, j:

$$\mathsf{rel}_x(M)_{i,j} = \begin{cases} M_{i,j} & \text{if } x_i \neq x \text{ and } x_j \neq x \\ (?, =) & \text{otherwise} \end{cases}$$

Inclusion. We note $M^1 \subseteq M^2$ iff $M^1_{i,j} \leq M^2_{i,j}$ for all $0 \leq i, j \leq n$.

Proposition 10. *Let M, M^1, M^2 be EDBMs in normal form, on set of clocks C. Then, (i) $\overrightarrow{[\![M]\!]} = \left[\!\left[\overrightarrow{M}\right]\!\right]$, (ii) $\overleftarrow{[\![M]\!]} = \left[\!\left[\overleftarrow{M}\right]\!\right]$, (iii) $[\![M^1 \cap M^2]\!] = [\![M^1]\!] \cap [\![M^2]\!]$, (iv) for all clock $x \in C$, $\mathsf{rel}_x([\![M]\!]) = [\![\mathsf{rel}_x(M)]\!]$ and (v) $[\![M^1]\!] \subseteq [\![M^2]\!]$ iff $M^1 \subseteq M^2$.*

Forward and backward analysis. We present now the forward and backward analysis algorithms adapted to ECA. From now on, we will consider an ECA $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$. We also let $\mathsf{Post}\left((q, v)\right) = \{(q', v') \mid \exists t, a : (q, v) \xrightarrow{t,a} (q', v')\}$ and $\mathsf{Pre}\left((q, v)\right) = \{(q', v') \mid \exists t, a : (q', v') \xrightarrow{t,a} (q, v)\}$ and we extend those operators to sets of states

```
ForwExact begin
    Let Visited = $\varnothing$ ; Let Wait = $\{(q_i, Z_0)\}$ ;
    while $Wait \neq \varnothing$ do
        Get and remove $(q, Z)$ from Wait ;
        if $q \in \alpha$ and $Z \subseteq Z_f$ then return Yes ;
        if there is no $(q, Z') \in$ Visited s.t. $Z \subseteq Z'$ then
            Visited := Visited $\cup \{(q, Z)\}$ ; Wait := Wait $\cup$ Post $((q, Z))$ ;
    return No ;

BackExact begin
    Let Visited = $\varnothing$ ; Let Wait = $\{(q, Z_f) \mid q \in \alpha\}$ ;
    while $Wait \neq \varnothing$ do
        Get and remove $(q, Z)$ from Wait ;
        if $q = q_i$ and $Z \subseteq Z_0$ then return Yes ;
        if there is no $(q, Z') \in$ Visited s.t. $Z \subseteq Z'$ then
            Visited := Visited $\cup \{(q, Z)\}$ ; Wait := Wait $\cup$ Pre $((q, Z))$ ;
    return No ;
```

Algorithm 2. The forward and backward algorithms

in the natural way. Moreover, given a set of valuations Z and a location q, we abuse notations and denote by (q, Z) the set $\{(q, v) \mid v \in Z\}$. Also, we let $\mathsf{Post}^*((q, Z)) = \bigcup_{n \in \mathbb{N}} \mathsf{Post}^n((q, Z))$ and $\mathsf{Pre}^*((q, Z)) = \bigcup_{n \in \mathbb{N}} \mathsf{Pre}^n((q, Z))$, where $\mathsf{Post}^0((q, Z)) = (q, Z)$ and $\mathsf{Post}^n((q, Z)) = \mathsf{Post}\left(\mathsf{Post}^{n-1}((q, Z))\right)$, and similarly for $\mathsf{Pre}^n((q, Z))$. The Post and Pre operators are sufficient to solve language emptiness for ECA:

Lemma 11 (adapted from [3], Lemma 1). *Let $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$ be an* ECA*, let $I = \{(q_i, v) \mid v$ is initial$\}$, and let $\overline{\alpha} = \{(q, v) \mid q \in \alpha$ and v is final$\}$. Then:*

$$\mathsf{Post}^*(I) \cap \overline{\alpha} \neq \varnothing \text{ iff } \mathsf{Pre}^*(\overline{\alpha}) \cap I \neq \varnothing \text{ iff } L(A) \neq \varnothing.$$

Let us show how to compute these operators on event-zones. Given a location q, an event-zone Z on $\mathbb{C}_\Sigma$, and an edge $e = (q, a, \psi, q') \in \delta$, we let:

$$\mathsf{Post}_e((q_1, Z)) = \begin{cases} \left(q', \left(\mathsf{rel}_{\overleftarrow{x_a}}\left(\mathsf{rel}_{\overrightarrow{x_a}}(\overrightarrow{Z} \cap (\overrightarrow{x_a} = 0)) \cap \psi\right)\right) \cap (\overleftarrow{x_a} = 0)\right) & \text{if } q_1 = q \\ \varnothing & \text{otherwise} \end{cases}$$

$$\mathsf{Pre}_e((q_1, Z)) = \begin{cases} \left(q, \overleftarrow{(\mathsf{rel}_{\overrightarrow{x_a}}(\mathsf{rel}_{\overleftarrow{x_a}}(Z \cap (\overleftarrow{x_a} = 0)) \cap \psi))} \cap (\overrightarrow{x_a} = 0)\right) & \text{if } q_1 = q' \\ \varnothing & \text{otherwise} \end{cases}$$

Then, it is easy to check that $\mathsf{Post}((q, Z)) = \cup_{e \in \delta} \mathsf{Post}_e((q, Z))$ and that $\mathsf{Pre}((q, Z)) = \cup_{e \in \delta} \mathsf{Pre}_e((q, Z))$. With the algorithms on EDBMs presented above, these definitions can be used to compute the Pre and Post of zones using their EDBM encodings. Remark that Pre and Post return *sets of event-zones* as these are not closed under union.

Let us now consider the ForwExact and BackExact algorithms to test for language emptiness of ECA, shown in Algorithm 2. In these two algorithms Z_0 denotes the zone $\bigwedge_{x \in \mathbb{H}_\Sigma} x = \bot$ containing all the possible initial valuations and Z_f denotes the

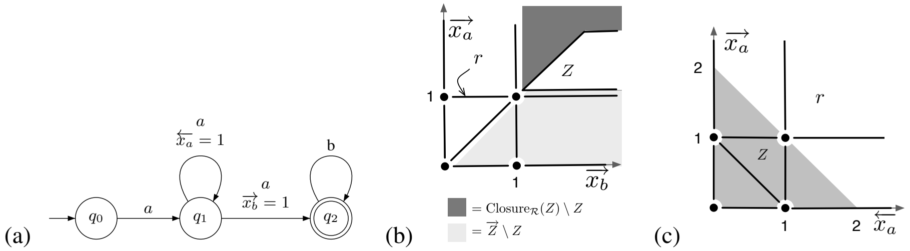


Fig. 3. (a): An ECA for which backward analysis does not terminate. (b) and (c): Examples for $\mathrm{Closure}_R$ and Approx_k.

zone $\bigwedge_{x\in\mathbb{P}_\Sigma} x = \bot$ representing all the possible final valuations. By Lemma 11, it is clear that ForwExact and BackExact are correct when they terminate. Unfortunately, Fig. 3 (a) shows an ECA on which the backward algorithm does not terminate. Since history and prophecy clocks are symmetrical, this example can be adapted to define an ECA on which the forward algorithm does not terminate either. Remark that in the case of timed automata, the forward analysis is not guaranteed to terminate, whereas the backward analysis *always terminates* (the proof relies on a bisimulation argument) [1].

Proposition 12. *Neither* ForwExact *nor* BackExact *terminate in general.*

Proof. We give the proof for BackExact, a similar proof for ForwExact can then be deduced by symmetry. Consider the ECA in Fig. 3 (a). Running the backward analysis algorithm from (q_2, Z_f), we obtain, after selecting the transition $e = (q_2, b, \mathtt{true}, q_2)$, the zone $Z_1 = \overrightarrow{x_a} = \bot \wedge \overleftarrow{x_b} = 0$. Then, the transition $e' = (q_1, a, \overrightarrow{x_b} = 1, q_2)$ is back-firable and we attain the zone $Z_2 = \overrightarrow{x_a} \geq 0 \wedge \overrightarrow{x_b} \geq 1 \wedge \overrightarrow{x_b} - \overrightarrow{x_a} = 1$. At this point the transition $e'' = (q_1, a, \overleftarrow{x_a} = 1, q_1)$ is back-firable, which leads to the zone $Z_3 = \overrightarrow{x_b} \geq 1 \wedge \overrightarrow{x_a} \geq 0 \wedge 0 \leq \overleftarrow{x_a} \leq 1 \wedge \overrightarrow{x_b} - \overrightarrow{x_a} \geq 1 \wedge \overrightarrow{x_b} + \overleftarrow{x_a} \geq 2$. The back-firing of the e'' transition can be repeated, and, by induction, after n iterations of the loop, the algorithm reaches the zone $Z^n = \overrightarrow{x_b} \geq n \wedge \overrightarrow{x_a} \geq 0 \wedge 0 \leq \overleftarrow{x_a} \leq 1 \wedge \overrightarrow{x_b} - \overrightarrow{x_a} \geq n \wedge \overrightarrow{x_b} + \overleftarrow{x_a} \geq n+1$. Thus, the condition of the **if** in line 14 is always fulfilled, and the algorithm visits an infinite number of zones, without reaching q_0. □

6 Future Work: Widening Operators

As said earlier, the zone-based *forward* analysis algorithm does not terminate either in the case of timed automata. To recover termination, *widening operators* have been defined. The most popular widening operator is the so-called k-approximation on zones [8]. Roughly speaking, it is defined as follows: in the definition of the zone, replace any constraint of the form $x_i \prec c$ or $x_i - x_j \prec c$, by respectively $x_i < \infty$ and $x_i - x_j < \infty$ if and only if $c > k$, and replace any constraint of the form $c \prec x_i$ or $c \prec x_i - x_j$, by respectively $k < x_i$ and $k < x_i - x_j$, if and only if $c > k$. Such an operator can be easily computed on DBMs, and is a standard operation implemented in several tools such as as UppAal [4] for more more than 15 years. Nevertheless, this operator has been widely discussed in the recent literature since Bouyer has pointed out several flaws in the proposed proofs of soundness [6]. Actually, the k-approximation is *sound* when the timed automaton contains *no diagonal constraints*. Unfortunately, k-approximation is *not sound* when the timed automaton contains diagonal constraints, and *no sound widening operator exists* in this case.

In [6], Bouyer identifies some subclasses of timed automata for which the widening operator is provably correct. The idea of the proof relies mainly on the definition of another widening operator, called the *closure by regions*, which is shown to be *sound*. The closure by regions of a zone Z, with respect to a set of regions $\mathcal{R}$ is defined as the smallest set of regions from $\mathcal{R}$ that have a non-empty intersection with Z, i.e. $\mathrm{Closure}_{\mathcal{R}}(Z) = \{r \in \mathcal{R} \mid Z \cap r \neq \varnothing\}$. Then, the proof concludes by showing that $\mathrm{Approx}_k(Z)$ is sound for some values of k (that are proved to exist) s.t.

$$Z \subseteq \mathrm{Approx}_k(Z) \subseteq \mathrm{Closure}_{\mathcal{R}}(Z). \tag{1}$$

In the perspective of bringing ECA from theory to implementation, *provably correct* widening operators are necessary, since neither the forward nor the backward algorithm terminate in general. We plan to adapt the k-approximation to ECA, and we believe that we can follow the general idea of the proof in [6]. However, the proof techniques will not be applicable in a straightforward way, for several reasons. First, the proof of [6] relies on the following property, which holds in the case of timed automata: for all zone Z and all location q: $\mathsf{Post}\left((q, \mathrm{Closure}_{\mathcal{R}}(Z))\right) \subseteq \mathrm{Closure}_{\mathcal{R}}(\mathsf{Post}\left((q, Z)\right))$. Unfortunately this is not the case in general with ECA. Indeed, consider the zone Z and the region r in Fig. 3 (b). Clearly, r is included in $\overrightarrow{\mathrm{Closure}_{\mathcal{R}}(Z)}$ but r is not included in $\mathrm{Closure}_{\mathcal{R}}(\overrightarrow{Z})$ (recall that prophecy clocks decrease with time elapsing). Moreover, the definition of the k approximation will need to be adapted to the case of ECA. Indeed, the second inclusion in (1) does not hold when using the k-approximation defined for timed automata, which merely replaces all constants $> k$ by ∞ in the constraints of the zone. Indeed, consider the event-zone Z defined by $\overleftarrow{x_a} + \overrightarrow{x_a} \leq 2$ in Fig. 3 (c), together with the set of regions $\mathcal{R} = \mathsf{Reg}\left(\mathbb{C}_{\{a\}}, 1\right)$. Clearly, with such a definition, the constraint $\overleftarrow{x_a} + \overrightarrow{x_a} \leq 2$ would be replaced by $\overleftarrow{x_a} + \overrightarrow{x_a} < \infty$, which yields an approximation that intersects with r, and is thus not contained in $\mathrm{Closure}_{\mathcal{R}}(Z)$. We keep open for future works the definition of a provably correct adaptation of the k-approximation for ECA.

References

1. Alur, R.: Timed automata. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 8–22. Springer, Heidelberg (1999)
2. Alur, R., Dill, D.: A Theory of Timed Automata. Theoretical Computer Science 126(2), 183–236 (1994)
3. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: a determinizable class of timed automata. Theoretical Computer Science 211(1-2), 253–273 (1999)
4. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: Proceedings of QEST 2006, pp. 125–126. IEEE Computer Society, Los Alamitos (2006)
5. Bellman, R.: Dynamic Programming. Princeton university press, Princeton (1957)

6. Bouyer, P.: Forward analysis of updatable timed automata. Formal Methods in System Design 24(3), 281–320 (2004)
7. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: A model-checking tool for real-time systems. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 546–550. Springer, Heidelberg (1998)
8. Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 313–329. Springer, Heidelberg (1998)
9. Di Giampaolo, B., Geeraerts, G., Raskin, J., Sznajder, N.: Safraless procedures for timed specifications. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 2–22. Springer, Heidelberg (2010)
10. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
11. Dima, C.: Kleene theorems for event-clock automata. In: Ciobanu, G., Păun, G. (eds.) FCT 1999. LNCS, vol. 1684, pp. 215–225. Springer, Heidelberg (1999)
12. D'Souza, D., Tabareau, N.: On timed automata with input-determined guards. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 68–83. Springer, Heidelberg (2004)
13. Geeraerts, G., Raskin, J.-F., Sznajder, N.: Event-Clock Automata: from Theory to Practice. Technical report., http://arxiv.org/abs/1107.4138
14. Raskin, J.-F., Schobbens, P.-Y.: The logic of event clocks: decidability, complexity and expressiveness. Automatica 34(3), 247–282 (1998)
15. Sorea, M.: Tempo: A model-checker for event-recording automata. In: Proceedings of RT-TOOLS 2001, Aalborg, Denmark (August 2001)
16. Tang, N., Ogawa, M.: Event-clock visibly pushdown automata. In: Nielsen, M., Kučera, A., Miltersen, P.B., Palamidessi, C., Tůma, P., Valencia, F. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 558–569. Springer, Heidelberg (2009)

On Construction of Safety Signal Automata for $MITL[\mathcal{U}, \mathcal{S}]$ Using Temporal Projections

Dileep Raghunath Kini[1], Shankara Narayanan Krishna[1], and Paritosh K. Pandya[2]

[1] Department of Computer Science & Engineering, IIT Bombay, Mumbai, India 400 076
{dileep,krishnas}@cse.iitb.ac.in
[2] School of Technology and Computer Science, Tata Institute of Fundamental Research,
Mumbai, India 400 005
pandya@tcs.tifr.res.in

Abstract. Construction of automata for Metric Temporal Logics has been an active but challenging area of research. We consider here the continuous time Metric temporal logic $\mathsf{MTL}[\mathcal{U}_I, \mathcal{S}_I]$ as well as corresponding signal automata. In previous works by Maler, Nickovic and Pnueli, the signal automaton synthesis has mainly addressed MTL under an assumption of bounded variability. In this paper, we propose a novel technique of "Temporal Projections" that allows easy synthesis of safety signal automata for continuous time $\mathsf{MITL}[\mathcal{U}_I, \mathcal{S}_I]$ over finite signals without assuming bounded variability. Using the same technique, we also give synthesis of safety signal automata for $\mathsf{MITL}[\mathcal{U}_I, \mathcal{S}_I]$ with bounded future operators over infinite signals. For finite signals, the Temporal Projections allow us to syntactically transform an MITL formula $\phi(Q)$ over a set of propositions Q to a pure past time MITL formula $\psi(P, Q)$ with extended set of propositions (P, Q) which is language equivalent "modulo temporal projection", i.e. $L(\phi) = L(\exists P.\boxdot\psi)$. A similar such transformation over infinite signals is also formulated for $\mathsf{MITL}[\mathcal{U}_I, \mathcal{S}_I]$ restricted to Bounded Future formlae where the Until operators use only bounded (i.e.non-infinite) intervals. It is straightforward to construct safety-signal-automaton for the transformed formula. We give complexity bounds for the resulting automaton. Our temporal projections are inspired by the use of projections by D'Souza *et al* for eliminating past in MTL.

1 Introduction

Logic automaton connection has proved to be influential in finding practical model checking technique for the verification of programs, as well as in theoretical studies on expressiveness and decidability of logics. For example, the well known Vardi-Wolper technique gives synthesis of a Buchi automaton recognizing the language of an LTL formula. By using this automaton as a synchronous monitor, the LTL formula can be model checked. This approach has been applied to several other logics too. However, extending this approach to timed logics has been challenging.

A prominent (linear) timed logic is $\mathsf{MTL}[\mathcal{U}_I, \mathcal{S}_I]$. Here the temporal modalities $\mathcal{U}_I$ and $\mathcal{S}_I$ are time constrained using a time interval I with integer end-points. The logic can be interpreted over different forms of time. For example, $\mathsf{MTL}[\mathcal{U}_I, \mathcal{S}_I]$ over timed words gives the so called pointwise MTL where as $\mathsf{MTL}[\mathcal{U}_I, \mathcal{S}_I]$ over timed state

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 225–239, 2011.

sequences (also called signals), gives the continuous timed MTL (see [1]). Moreover, the timed behaviour may be infinite (extending up to infinity in time) or finite. In this paper, we confine ourselves to continuous timed MTL. Logics MTL over finite and infinite behaviours are respectively denoted as $\mathsf{MTL}_{\mathsf{fin}}$ and $\mathsf{MTL}_{\mathsf{inf}}$.

The expressiveness and decidability properties of the temporal logic vary according to the nature of time assumed, permitted operators and permitted time intervals. Full $\mathsf{MTL}[\,\mathcal{U}_I,\ \mathcal{S}_I]$ is undecidable for both finite and infinite behaviours. Alur, Feder and Henzinger [2] proposed $\mathsf{MITL}[\mathcal{U}_I,\ \mathcal{S}_I]$ as a restriction of $\mathsf{MTL}[\mathcal{U}_I,\ \mathcal{S}_I]$ where singular (or punctual) intervals of the form $[l, l]$ are disallowed in formulae. Alur, Feder and Henzinger established the EXPSPACE satisfiability of $\mathsf{MITL}[\,\mathcal{U}_I]$ by constructing a tableaux for the formula. However, the proposed construction for $\mathsf{MITL}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ is complex and not very practicable. Hence, several subsequent papers have addressed simpler techniques for the construction of signal automata for various fragments of continuous time $\mathsf{MTL}[\,\mathcal{U}_I,\ \mathcal{S}_I]$ [6,7,8]. Thus, Maler et al [6] exhibited synthesis of deterministic signal automata for the purely past time logic $\mathsf{MITL}_{\mathsf{fin}}[\mathcal{S}, \diamondminus_I]$. Extending this, Maler et al [7] gave a construction of a deterministic signal automaton $A(\phi, k)$ for a bounded future formula $\mathsf{MTL}[\,\mathcal{U}_J,\ \mathcal{S}_I]$ and a variability index k. The automaton accepts exactly those k-variable signals which satisfy the formula ϕ. A signal is called k-variable if it does not change more than k times in any unit interval.

In this paper, we consider a novel technique for the synthesis of $\mathsf{MITL}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ formula automaton without any restriction of bounded variability. Note that this logic $\mathsf{MITL}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ is equivalent to the more traditional $\mathsf{MITL}[\,\mathcal{U}_I,\ \mathcal{S}_I]$. Our main results are as follows.

- For $\mathsf{MITL}_{\mathsf{fin}}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ formula (over finite signals) we construct a safety signal automaton which accepts a signal iff the signal satisfies the formula.
- For BMITL (over infinite signals) which consists only of formulae with bounded future, we construct a safety signal automaton which accepts a signal iff the signal satisfies the formula.

In both cases, the automaton construction uses a novel technique called *temporal projections*. A formula $\boxdot\phi$ holds for a behaviour if it holds at every point in the behaviour. A formula ϕ over a set of propositions Q is called "equivalent modulo projection" to a formula ψ over a set of propositions $P \cup Q$ provided $\phi \equiv \exists P \psi$. The composite operator $\exists P \boxdot$ is named temporal projection.

For finite signals, the Temporal Projections allow us to syntactically transform an $\mathsf{MITL}_{\mathsf{fin}}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ formula $\phi(Q)$ over a set of propositions Q to a pure past time $\mathsf{MITL}_{\mathsf{fin}}[\mathcal{S}, \diamondminus_I]$ formula $\psi(P, Q)$ with extended set of propositions (P, Q) which is language equivalent "modulo temporal projection", i.e. $L(\phi) = L(\exists P \boxdot \psi)$. Our temporal projections are inspired by the use of projections by D'Souza *et al* [3] for eliminating past in MTL. They allow us to state future requirements in terms of past requirements by shifting the time point of reference. A similar such transformation over infinite signals is also formulated for BMITL where the future operators use only bounded (i.e.non-infinite) intervals. Thus, temporal projections allow us to reduce $\mathsf{MITL}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ formulas to a purely past time formula.

In this paper, we consider timed state sequences in their full generality. We also consider the full logic MITL without any restrictions. This must be contrasted with [6,7] where some restrictions are placed on the timed state sequences as well as the intervals permitted in formulae. Dealing with this generality forces us to extend the signal automaton definition of Maler *et al* to automata which can distinguish between signals with open/closed/half-open/singular intervals. In this richer setting, the notion of *deterministic* signal automata and their closure properties requires some care in formulation. Using these automata, we first adapt the synthesis of deterministic signal automata for pure past time $\mathsf{MITL}[\mathcal{S}, \diamondminus_I]$ given by Maler *et al* [7] to all finitely variable signals. Thus, in Section 6, we give the synthesis of language equivalent signal-safety-automaton for a formula $\boxdot\psi$ with past time $\mathsf{MITL}[\mathcal{S}, \diamondminus_I]$ formula ψ. A projection operation on the resulting automaton eliminates the added witness propositions P in formula $\exists P \boxdot \psi$ by introducing nondeterminism. Thus, we obtain a rather simple technique for synthesizing signal automata for full $\mathsf{MITL}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ by first reducing it to temporal projection of pure past time MITL. The resulting automata are exponential in the size of the formula.

2 Syntax and Semantics of MTL

The time domain that is considered is $\mathbb{R}_{\geq 0}$, the set of non negative reals. Intervals as usual are convex subsets of $\mathbb{R}_{\geq 0}$, they may be open or closed from either side and they may be bounded or unbounded. Interval arithmetic like $t + I$ is used to indicate the set $\{t' | t' - t \in I\}$ while $t - I$ is used to indicate $\{t' | t - t' \in I\}$.

Continuous time MTL formulae are evaluated over *timed state sequences* (TSS) (also called *finitely variable signals*) which act as models of behaviours for timed systems. We use the terms *TSS* or *signal* or *behaviour* interchangeably while talking about models for timed systems. Let P be the set of atomic propositions. A *TSS* (or *signal*) is a pair $\tau = (\bar{s}, \bar{I})$, where $\bar{s}$ is a finite or infinite sequence $s_0, s_1, \ldots$ of subsets of P and $\bar{I}$ is a corresponding sequence $I_0, I_1, \ldots$ of intervals satisfying:

- I_0 is of the form $[0, r]$ or $[0, r)$ where $r \geq 0$
- $\forall i \geq 0$ intervals I_i and I_{i+1} are disjoint and $I_i \cup I_{i+1}$ is an interval. Let $Dom(\tau) = \cup_i I_i$.
- The TSS τ is called *infinite* if $\cup_i I_i = \mathbb{R}_{\geq 0}$. We require that τ is time divergent, i.e. if $\bar{s}$ is an infinite sequence then τ is necessarily infinite.
- TSS τ is called *finite* if $Dom(\tau) = [0, M]$ for some real number M. In this case, we require that $\bar{I}$ is finite and the last interval I_n is of the form $(r, M]$ or $[r, M]$.

The TSS $\tau = (\bar{s}, \bar{I})$ denotes a finitely variable function which takes value s_i throughout the interval I_i. A function over reals is finitely variable if it has only finitely many discontinuities in any finite interval. Hence, will use $\tau(t) = s_i$ to denote that $t \in I_i$.

The syntax of $\mathsf{MTL}[\,\mathcal{U}_I,\ \mathcal{S}_I]$ over the set of propositions P is given by

$$\varphi ::= p(\in P) \mid BP \mid EP \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\,\mathcal{U}_I\varphi \mid \varphi\,\mathcal{S}_I\varphi$$

where I is an interval with endpoints in $\mathbb{N}_0 \cup \infty$. Note that I can be open, closed, half-open, singleton (i.e.$[c, c]$) or unbounded (i.e. $\langle c, \infty)$). We use $\langle$ to range over $[, ($ and $\rangle$ to range over $],)$.

Let τ be a (finite or infinite) TSS over P and $t \in Dom(\tau)$. The pair (τ, t) is called a *model*. Then, the semantics of continuous time MTL is given by

$$\begin{array}{lll}
(\tau,t) \vDash p & \leftrightarrow & p \in \tau(t) \\
(\tau,t) \vDash BP & \leftrightarrow & t = 0 \\
(\tau,t) \vDash EP & \leftrightarrow & Dom(\tau) = [0,M] \wedge t = M \\
(\tau,t) \vDash \neg\psi & \leftrightarrow & (\tau,t) \nvDash \psi \\
(\tau,t) \vDash \psi_1 \vee \psi_2 & \leftrightarrow & (\tau,t) \vDash \psi_1 \text{ or } \tau,t \vDash \psi_2 \\
(\tau,t) \vDash \psi_1\, \mathcal{U}_I \psi_2 & \leftrightarrow & \exists t' \in t + I \wedge t' > t, (\tau,t') \vDash \psi_2 \text{ and } \forall t'' \in (t,t'), (\tau,t'') \vDash \psi_1 \\
(\tau,t) \vDash \psi_1\, \mathcal{S}_I \psi_2 & \leftrightarrow & \exists t' \in t - I \wedge t' < t, (\tau,t') \vDash \psi_2 \text{ and } \forall t'' \in (t',t), (\tau,t'') \vDash \psi_1
\end{array}$$

Following the anchored interpretation of temporal logic, let $\tau \models \psi$ iff $\tau, 0 \models \psi$. Let $L(\psi) = \{\tau \mid \tau, 0 \models \psi\}$. It is very useful to define strong satisfaction of a formula: let $\tau \models \boxdot\psi$ iff $\tau, t \models \psi$ for all $t \in Dom(\tau)$. Finally, let $L(\boxdot\psi) = \{\tau \mid \tau \models \boxdot\psi\}$. The strong satisfaction operator will play a crucial role in this paper.

Derived Operators. Define untimed modalities $\mathcal{U} \stackrel{\text{def}}{=} \mathcal{U}_{(0,\infty)}$, $\mathcal{S} \stackrel{\text{def}}{=} \mathcal{S}_{(0,\infty)}$, $\Diamond \stackrel{\text{def}}{=} \Diamond_{(0,\infty)}$ and $\diamondminus \stackrel{\text{def}}{=} \diamondminus_{(0,\infty)}$. Also, let unary modalities $\Diamond_I \stackrel{\text{def}}{=} True\, \mathcal{U}_I \psi$ and $\diamondminus_I \psi \stackrel{\text{def}}{=} True\, \mathcal{S}_I \psi$. Define $\Box_I \psi \stackrel{\text{def}}{=} \neg\Diamond_I\neg\psi$ and $\boxminus_I \psi \stackrel{\text{def}}{=} \neg\diamondminus_I\neg\psi$. Let $\nearrow \psi \stackrel{\text{def}}{=} \psi\, \mathcal{U} \psi$. This holds at any point where ψ is continuously true in its small right neighbourhood. Alternately, define $\nearrow \psi \stackrel{\text{def}}{=} \psi\, \mathcal{U}_{(0,1)} \psi$ using a bounded modality. Similarly, we can define $\nwarrow \psi$.

Sublogics. The sublogic (syntactic subset) of MTL where singular (punctual) intervals of the form $[c, c]$ are disallowed in the formulae is called MITL. We use MTL$[OPLIST]$ and MITL$[OPLIST]$ to denote subset of formulae where only the temporal operators from $OPLIST$ are used. For example MITL$[\mathcal{S}, \diamondminus_I]$ denotes the past-time fragment of MITL. When formulae are interpreted only over finite (or infinite) signals we denote this by MTL$_{\mathsf{fin}}$ (or MTL$_{\mathsf{inf}}$).

Transformations. All forms of open/closed/half-open intervals are allowed in the syntax of MITL formulae. The following equivalences show that it is sufficient to use formulae with only the open interval and the reduction causes only linear blowup in size.

Lemma 1. – *We have,* $\Diamond_{[0,r)} p \;\leftrightarrow\; p \vee \Diamond_{(0,r)} p$

– *For* $l \neq 0$, *we have* $\Diamond_{[l,r)} p \;\leftrightarrow\; \Diamond_{(l,r)} p \vee \Box_{(l-1,l)} \Diamond_{(0,1)} (p \wedge (\nearrow \neg p))$

$$\Diamond_{(l,r]} p \;\leftrightarrow\; \Diamond_{(l,r)} p \vee \Box_{(r-1,r)} \Diamond_{(0,1)} (p \wedge (\nwarrow \neg p))$$

Next, we show that it is possible to remove the modalities $\mathcal{U}_I$ and $\mathcal{S}_I$. An equivalent formula using only the untimed modalities $\mathcal{U}$, $\mathcal{S}$ and timed unary modalities $\Diamond_I$,$\diamondminus_I$ can be constructed with only linear blowup in size. Hence, in the rest of this paper, in place of MITL$[\mathcal{U}_I, \mathcal{S}_I]$ we shall confine ourselves to MITL$[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ where only open intervals I are used.

Lemma 2. $p\,\mathcal{U}_{(l,r)} q \;\leftrightarrow\; \Diamond_{(l,r)} q \wedge \Box_{(0,l]} (p \wedge (p\,\mathcal{U} q))$

Now we define the notion of *reflection* for finite behaviours and MTL$[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ formulae and show that formula satisfaction is preserved under reflection. Let τ be a finite signal with $Dom(\tau) = [0, d]$. Then, The reflection of τ denoted by $\hat{\tau}$ is a finite signal with $Dom(\hat{\tau}) = [0, d]$ such that $p \in \tau(t)$ iff $p \in \hat{\tau}(\hat{t})$ where $\hat{t} = d - t$.

The reflection of a formula $\alpha \in \mathsf{MTL}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ denoted by $\hat{\alpha}$ is a formula obtained by exchanging future and past operators, i.e. $(\mathcal{U} \leftrightarrow \mathcal{S})$, $(\Diamond_I \leftrightarrow \diamondminus_I)$ and $(BP \leftrightarrow EP)$.

Proposition 1. *For any finite behaviour τ $(\tau, t) \models \alpha$ iff $(\hat{\tau}, \hat{t}) \models \hat{\alpha}$ and hence $\tau \models \boxdot\alpha$ iff $\hat{\tau} \models \boxdot\hat{\alpha}$.* □

3 Temporal Projections and $\mathcal{S}$ Operator Elimination

Let P be a set of propositions, and let τ' be a signal over $P \cup P'$, where P' is a set of propositions disjoint from P. Then $\tau' \upharpoonright P$ is a signal over P s.t. $(\tau' \upharpoonright P)(t) = \tau'(t) \cap P$ forall $t \in Dom(\tau)$. Given τ over P a signal τ' over $P \cup P'$ is called P'-extension of τ provided $\tau' \upharpoonright P = \tau$.

Given a formula ψ_i over P and a fresh proposition q_i, the equivalence $q_i \Leftrightarrow \psi_i$ is called a temporal definition and q_i is called its witness. Let X be a given conjunction of temporal definitions. Let the set of its witnesses be P'. Then, a P'-extension τ' of τ is called *canonical* with respect to X provided $\tau' \models \boxdot X$. For every τ (over P) there is a unique canonical extension τ' w.r.t. X.

Let ψ be MTL formula over propositions $P \cup P'$ and let τ be a signal over P. Define $\tau \models \exists P' \boxdot \psi$ iff $\tau' \models \boxdot\psi$ for some P'-extension τ' of τ. Hence, let $L(\exists P' \boxdot \psi) = \{\tau' \upharpoonright P \mid \tau' \models \boxdot\psi\}$.

Temporal projections $\exists P' \boxdot \psi$ are quite powerful. Given a formula ϕ (over P) of logic L_1, we can often find a formula ψ (over $P \cup P'$) of a much simpler/desirable logic L_2 such that $L(\phi) = L(\exists P' \boxdot \psi)$. We say that ϕ is equivalent modulo temporal projection to ψ. For example, we can flatten a formula ϕ (over Q) to formula ψ by introducing a fresh proposition p_i for each subformula of ϕ such that $\tau, 0 \models \phi$ iff $\tau \models \exists p_1, \ldots, p_n \boxdot \psi$. We illustrate this by an example below. Full details can be found in [3].

Example 1. Let $\phi = (q_1\, \mathcal{U}_I(q_2\, \mathcal{S} q_3)) \vee \diamondminus_J q_4$. Define ψ as conjunction of formulae $(p_1 \Leftrightarrow \diamondminus_J q_4)$ and $(p_2 \Leftrightarrow (q_2\, \mathcal{S} q_3))$ and $(p_3 \Leftrightarrow (q_1\, \mathcal{U}_I p_2))$ and $(BP \Rightarrow (p_3 \vee p_1))$. Then, $L(\phi) = L(\exists p_1, \ldots p_3. \boxdot \psi)$.

In the next two sections, we shall give several other examples of equivalence modulo temporal projections.

$\mathcal{S}$ elimination. D'Souza et al showed how to eliminate untimed $\mathcal{S}$ using $\mathcal{U}$ and temporal projections [3]. We adapt their construction and make it apply uniformly to both finite and infinite signals. Let $rd(\psi)$ be defined as $\psi \wedge (\nearrow \neg\psi)$ and let

$$\alpha(\psi, \mu) \equiv (\psi\, \mathcal{U}((\neg\psi) \vee rd(\psi) \vee EP)) \Rightarrow ((\psi \wedge \mu)\, \mathcal{U}(((\neg\psi) \vee rd(\psi) \vee EP) \wedge \mu))$$

Note that $\alpha(\psi, \mu)$ holds at a point t provided for all $t' > t$, if ψ holds in (t, t') then μ holds in $(t, t']$.

$$\nu = (BP \Rightarrow \neg r) \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5$$

such that

1. $\varphi_1 : \alpha(p \wedge q, r)$
2. $\varphi_2 : ((r \wedge p) \vee q) \Rightarrow \alpha(p \wedge \neg q, r)$
3. $\varphi_3 : \neg((r \wedge p) \vee q) \Rightarrow \alpha(p \wedge \neg q, \neg r)$
4. $\varphi_4 : \alpha(\neg p \wedge \neg q, \neg r)$
5. $\varphi_5 : \alpha(\neg p \wedge q, \neg r)$

Lemma 3. [3] *Let $\nu \in \mathsf{MTL}[\mathcal{U}, BP, EP]$ be as constructed above. Then, for any (finite or infinite) signal τ, we have $\tau \models \boxdot\nu$ iff $\tau \models \boxdot(p\mathcal{S}q \Leftrightarrow r)$.* □

Applying the reflection property (proposition 1) to above lemma, we can also eliminate $\mathcal{U}$ using $\mathcal{S}$ over finite behaviours. Let $\hat{\nu}$ be the reflection of formula ν given above.

Corollary 1. *For a finite signal τ, we have $\tau \models \boxdot\hat{\nu}$ iff $\tau \models \boxdot(p\,\mathcal{U}q \Leftrightarrow r)$. Note that $\hat{\nu} \in \mathsf{MTL}_{\mathsf{fin}}[\mathcal{S}, BP, EP]$.*

D'Souza et al [3] also showed how to eliminate $\diamondminus_I$ operator using only the future operators and temporal projections. We can generalize their proof to give a uniform $\diamondminus_I$ elimination for finite and infinite signals. Moreover, in contrast with [3], we do not have any prefix where the elimination is not applicable. This is captured in the following theorem whose proof can be found in the full version of the paper.

Theorem 1. *For every $\varphi \in \mathsf{MITL}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ over P we can construct $\psi_{fut} \in \mathsf{MITL}[\mathcal{U}, \Diamond_I]$ over $P' \cup P$ such that $L(\phi) = L(\exists P'. \boxdot\, \psi_{fut})$.*

4 Eliminating Future Modality $\Diamond_I$

In this section, we look at eliminating $\Diamond_I$ from $\mathsf{MITL}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ formula. This result, coupled with the $\mathcal{U}$ elimination of Corollary 1 helps us to obtain a pure past formula starting from a formula in $\mathsf{MITL}_{\mathsf{fin}}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$. In previous works, [7] has eliminated $\Diamond_I$ from formlae in $\mathsf{MTL}_{\mathsf{inf}}[\mathcal{U}, \Diamond_I]$ with bounded intervals I to obtain equivalent past formulae.

Theorem 2. *For every $\varphi \in \mathsf{MITL}_{\mathsf{fin}}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ over P, we can construct $\psi_{past} \in \mathsf{MITL}_{\mathsf{fin}}[\mathcal{S}, \diamondminus_I]$ over $P' \cup P$ such that $L(\varphi) = L(\exists P' \boxdot\, \psi_{past})$.*

Proof. First we eliminate $\mathcal{U}$ using Corolloary 1. Then to handle $\Diamond_I$ we construct a formula that gets rid of $\Diamond_I$ using only $\diamondminus_I$, $\nwarrow$ and $\mathcal{S}$. As seen in Lemma 1, it is sufficient to consider formulae with only open intervals.

First we consider the case when I is of the form (l, ∞). Let the temporal definition

$$X = (q \Leftrightarrow \Diamond_I p) \wedge (p' \Leftrightarrow \Diamond p)$$

Here p', q are the witness propositions. We construct $\nu_p \in \mathsf{MITL}_{\mathsf{fin}}[\mathcal{S}, \diamondminus_I, \mathcal{U}]$ and we claim that $\tau' \models \boxdot X$ iff $\tau' \models \boxdot\nu_p$. Let $\nu_p = \varphi_1 \wedge \varphi_2 \wedge \varphi_3$, where

$$\varphi_1 : (p \Rightarrow \boxminus p') \wedge (EP \Rightarrow ([\neg p' \wedge \boxminus\neg p')] \vee [\neg p' \wedge (\neg p'\,\mathcal{S}(p \wedge \neg p'))])$$
$$\varphi_2 : \neg p' \Rightarrow \boxminus_{[0,l]}\neg q$$
$$\varphi_3 : (p \Rightarrow \boxminus_{(l,\infty)} q)$$

First we show that $\tau' \models \boxdot X$ implies $\tau' \vDash \boxdot \nu_p$:

1. φ_1: If p occurs at a point, then by the definition of canonical extension, p' should be true for all points before it, and p' cannot become true at a point unless p occurs once later.
2. φ_2 : If p does not become true in the future, then $\Diamond_{(l,\infty)}p$ cannot become true for at least l length of time.
3. φ_3: If p occurs at t then $\Diamond_I p$ will be true throughout $t - I$. As $I = (l, \infty)$, we have $\boxminus_{(l,\infty)} q$ holds at t.

Now we prove the converse that $\tau' \vDash \boxdot \nu_p$ implies $\tau' \models \boxdot X$: The first two points below establish the relationship between p and p' while the last two establish the relationship between p and q.

1. Let $(\tau', t) \vDash \Diamond p$. Then, the formula $p \Rightarrow \boxminus p'$ ensures p' is true at t.
2. Let $(\tau', t) \nvDash \Diamond p$. Then, $EP \Rightarrow ([\neg p' \wedge \boxminus \neg p')] \vee [\neg p' \wedge (\neg p' \, \mathcal{S}(p \wedge \neg p'))])$ ensures that p' is false from EP leftward up to the point where p became true for the last time from BP.
3. Let $(\tau', t) \vDash \Diamond_I p$. Then φ_3 ensures q is true at t.
4. Let $(\tau', t) \nvDash \Diamond_I p$. p does not occur anywhere after $t + l$, so p' is false at $t + l$ and then φ_2 ensures q is false at t.

Next, we consider the case when the interval is bounded i.e I is of the form (l, r). Let the temporal definition

$$X = (q \Leftrightarrow \Diamond_I p) \wedge (p' \Leftrightarrow \Diamond p).$$

We construct $\nu_p \in \mathsf{MITL_{fin}}[\mathcal{S}, \diamondminus_I]$ and we claim that $\tau' \vDash \boxdot X$ iff $\tau' \vDash \boxdot \nu_p$.
Define $\nu_p \overset{\text{def}}{=} \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5 \wedge \varphi_6 \wedge \varphi_7$, where

$$\begin{aligned}
&\varphi_1 : (p \Rightarrow \boxminus_{(0,\infty)} p') \wedge EP \Rightarrow ([\neg p' \wedge \boxminus \neg p')] \vee [\neg p' \wedge (\neg p' \, \mathcal{S}(p \wedge \neg p'))]) \\
&\varphi_2 : (\neg p' \Rightarrow \boxminus_{[0,l]} \neg q) \\
&\varphi_3 : (p \Rightarrow \boxminus_{(l,r)} q) \\
&\varphi_4 : \neg q \Rightarrow [\boxminus \neg q \vee ((\neg q \, \mathcal{S}(\boxminus_{(0,r-l)} q \wedge \neg q))] \\
&\varphi_5 : (q \Rightarrow (q \mathcal{S} \neg q)) \vee \boxminus q \\
&\varphi_6 : [\boxminus_{(0,r-l)} \neg p] \Rightarrow \diamondminus_{(l,r]} \neg q \vee \diamondminus_{(0,r)} BP \\
&\varphi_7 : [\boxminus_{(l,r)} q \wedge \diamondminus_{[l,r)} \neg q] \Rightarrow [p \vee (\nwarrow p)]
\end{aligned}$$

First we show that $\tau' \models \boxdot X$ implies $\tau' \vDash \boxdot \nu_p$:

1. φ_1 φ_2 φ_3 : same as in the case (l, ∞)
2. φ_4: We show that maximal q intervals are right open and of length at least $r - l$. If maximal q intervals were not right open, then there is a point t where q holds, and q does not hold in the immediate right neighbourhood of t. So, for any small $\epsilon > 0$, $\neg q$ holds at $t + \epsilon$. Using the fact that q is true at t and $\neg q$ is true at $t + \epsilon$ for any small ϵ, we have $\Diamond_{(l,r)} p$ true at t and $\Box_{(l,r)} \neg p$ at $t + \epsilon$. Let p hold at $t' \in (t + l, t + r)$. Thus, $t' > t + l$. It is possible to pick a sufficiently small ϵ such that $t' > t + l + \epsilon$. This contradicts the fact that $\Box \neg p$ holds in $(t + l + \epsilon, t + r + \epsilon)$.

To show that the length of the maximal q-intervals is at least $r - l$. Consider a maximal q-interval (t_1, t_2) with $t_2 - t_1 < r - l$. Let $I_i = t_i + (l, r)$ for $i = 1, 2$. By definition $\Diamond p$ holds in both I_1, I_2. As $t_2 - t_1 < r - l$, $(t_2 + l, t_1 + r) \subset I_1, I_2$. Pick a point t_3 in (t_1, t_2), and let $I_3 = t_3 + (l, r)$. Then $I_3 \subset I_1 \cup I_2$. We know q holds at t_3, hence $\Diamond p$ holds in I_3. The fact that (t_1, t_2) is a maximal interval implies that we have $\Box p$ in $(t_1 - \epsilon + l, t_1 - \epsilon + r)$ as well as $(t_2 + \epsilon + l, t_2 + \epsilon + r)$ for any small ϵ. This contradicts the fact that $\Diamond p$ holds in I_3.

3. φ_5: Maximal q intervals are left open. Similar to above.
4. φ_6: Suppose $\boxminus_{(0,r-l)}\neg p$ holds at t. Then $\neg q$ holds at $t - r$, and hence $\diamondminus_{(l,r]}\neg q$ holds at t. If $t - r$ does not exist (for instance, if $t = r - l$) then we say $\diamondminus_{(0,r)}BP$.
5. φ_7: Suppose the LHS of the implication φ_7 is true at t. Then $\Box\neg p$ holds throughout $(t, t + (r - l))$. If $\nwarrow p$ and p are both not true at t, then then we can find a small enough $\epsilon > 0$ such that $\Box\neg p$ holds in $(t - \epsilon, t]$. Since q holds throughout $(t - r, t - l)$, we can pick ϵ such that q holds at $t - l - \epsilon$. This means, $\Diamond p$ holds in $(t - \epsilon, t + (r - l) - \epsilon)$, which is a contradiction.

Now, we prove the converse, i.e. $\tau' \vDash \Box\nu_p$ implies $\tau' \models \Box X$: φ_1 ensures that the relationship between p and p' as in the definition of canonical-extension is met. Now using the rest of the formulae we prove the relationship between p and q:

If $(\tau', t_1) \vDash \Diamond_{(l,r)}p$ then φ_3 ensures $\tau', t_1 \vDash q$.

Now assume $(\tau', t_1) \nvDash \Diamond_{(l,r)}p$. Let us denote by EP the end point of τ'.

case $EP - t_1 < r$: When $EP - t_1 > l$, it is easy to see that φ_2 ensures $\tau', t_1 \vDash \neg q$. If $EP - t_1 \leq l$, then using φ_1, we have $\neg p'$ holds good at EP, and then φ_2 gives that $\neg q$ is true at t_1.

case $EP - t_1 \geq r$: suppose $\tau', t_1 \vDash q$, then there exists a point t_2 which is the earliest point after t_1 at which $\neg q$ holds. Such a point does exist because q-intervals are right open and $\neg q$ holds at EP. Now $EP - t_2 \geq l$ because $\neg p'$ is true at EP and φ_2 says $\neg q$ holds in $[EP - l, EP]$.

Applying φ_7 at $t_2 + l$ we obtain that at $t_2 + l$, $p \vee (\nwarrow p)$ should hold. Since $t_1 < t_2$ we have $t_1 + l < t_2 + l$. By assumption, $\tau', t_1 \nvDash \Diamond_{(l,r)}p$ which implies p cannot hold anywhere between $t_1 + l$ and $t_1 + r$. Putting the last two lines together, we get that $t_1 + r < t_2 + l$. Now consider the earliest point on or after $t_1 + r$ where $p \vee (\nearrow p)$ is true, call it t_3. Such a point exists because we know there is a point namely $t_2 + l > t_1 + r$ at which $p \vee (\nwarrow p)$ holds. Now we have $t_2 + l \geq t_3 > t_1 + r$. Applying φ_6 at t_3 we get $\tau, t_3 - r \vDash \neg q$, but we also have $t_1 < t_3 - r \leq t_2 - (r - l)$ and we know q holds throughout $[t_1, t_2)$. This is a contradiction. Hence, it cannot be the case that $\tau, t_1 \vDash q$.

To prove the statement of our theorem, let $\varphi \in \mathsf{MITL_{fin}}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ be the given formula. First we replace each $\Diamond_I p$ subformulae in φ with new witness propositions q from P'. Call this formula ψ. Then we add the conjuncts $W_p = [(q \Leftrightarrow \Diamond_I p) \wedge (\Diamond p \Leftrightarrow p')]$ to ψ. We need to remove the conjuncts W_p from ψ. Use $\bigwedge_{p \in P} \nu_p$ to replace this, to get the formula $\psi \wedge \bigwedge_{p \in P} \nu_p$. Our final formula ψ_{past} will be $(BP \Rightarrow \psi) \wedge \bigwedge_{p \in P} \nu_p$. □

A result similar to Theorem 2 is proved by Maler *et al* in [7]. They showed that a formula $\phi \in \mathsf{MTL_{inf}}[\mathcal{U}, \Diamond_I]$ with bounded intervals can be transformed into $\phi' \in \mathsf{MITL_{inf}}[\mathcal{S}, \diamondminus_I]$ such that $L(\phi) = L(\phi')$. Our result differs in the following ways:

1. Maler *et al* [7] crucially uses punctuality to make the transformation from future to past, where as in our case punctuality is disallowed, and hence our method differs significantly.
2. The method of [7] would extend to finite behaviour as well but if we try to eliminate unbounded $\mathcal{U}$ using the same technique one would a priori have to know the length of the behaviour to construct the formula

It should be noted that this method of interchanging future and past in Theorem 2 will not work in the case of infinite behaviours because the time domain in that case is asymmetric with respect to future and past. So in order to eliminate future in the infinite domain, we will have to follow a different method as shown in the next theorem.

4.1 Eliminating Bounded Future over Infinite signals

Let BMITL denote $\mathsf{MITL}_{\mathsf{inf}}[\Diamond_J, \mathcal{S}, \diamondminus_I]$ with bounded future intervals. That is, the intervals J appearing in $\Diamond_J$ are bounded (i.e. not infinite).[1]

Lemma 4. *For any infinite signal τ we have $\tau \models \boxdot(\nearrow p \Leftrightarrow q)$ iff $\tau \models \boxdot(\hat{\alpha}(p, q) \wedge \hat{\alpha}(\neg p, \neg q))$.*

Theorem 3. *For every $\phi \in \mathsf{BMITL}$ over P, we can construct $\psi_{past} \in \mathsf{MITL}_{\mathsf{inf}}[\mathcal{S}, \diamondminus_I]$ over $P' \cup P$ such that $L(\varphi) = L(\exists P' \boxdot \psi_{past})$.*

Proof. Consider the temporal definition

$$X = (q \Leftrightarrow \Diamond_I p) \wedge (p' \Leftrightarrow \Diamond p)$$

As in Theorem 2, we construct a formula ν_p that characterizes the canonical extensions. Let

$$\begin{aligned}
&\varphi_1 : (p \Rightarrow \boxminus_{(l,r)} q)\\
&\varphi_2 : \neg q \Rightarrow ((\neg \boxminus_{(0,r-l)}\, q) \Rightarrow \hat{\alpha}(\neg \boxminus_{(0,r-l)}\, q, \neg q))\\
&\varphi_3 : q \Rightarrow ((q\, \mathcal{S} \neg q) \vee \boxminus q)\\
&\varphi_4 : (\boxminus_{(0,r-l)} \neg p \wedge \boxminus_{(r,2r-l)} q \wedge \neg \diamondminus_{[0,r)} BP) \Rightarrow \diamondminus_{[r,2r-l)}(\neg q)\\
&\varphi_5 : (\boxminus_{(l,r)} q \wedge \diamondminus_{(l,r]} \neg q) \Rightarrow p \vee (\nearrow p)
\end{aligned}$$

Define $\nu_p \overset{\text{def}}{=} \varphi_1 \wedge \ldots \wedge \varphi_5$.

An important point to note here is that although in general it may not be possible to eliminate $\mathcal{U}$ in the infinite time domain, it is definitely possible to do so in the special case when it occurs in the form of $\nearrow p$, as seen in Lemma 4.
Let τ' be a canonical extension of τ. Then $\varphi_1 \wedge \ldots \varphi_5$ are satisfied at all points of τ'.

1. φ_1: If p occurs at t then $\Diamond_I p$ will be true throughout $t - I$ because $\forall t' \in t - I \;\; \tau, t \models p$ implies $\forall t' : t \in t' + I \;\; \tau, t \models p$. Hence by semantics of $\Diamond_I$ we have $\tau, t' \models \Diamond_I p$ and τ' is a canonical extn which implies $\tau', t' \models q$. Hence φ_1 holds.

[1] Extending sublogic BMITL with bounded $\mathcal{U}_J$ does not increase its expressive power as shown in the full version of the paper.

2. φ_2: Maximal q intervals are right open and are either atleast of length $r - l$ or interrupted by BP.
   We first show why maximal q-intervals should be right open. If not, then there is a point t where q holds, and q does not hold in the immediate right neighbourhood of t. Pick a sufficiently small ϵ such that at $t + \epsilon$, $\neg q$ holds. Using the fact that q is true at t and $\neg q$ is true at $t + \epsilon$ for any small ϵ, and by virtue of τ' being a canonical extn, we have $\Diamond_{(l,r)}p$ is true at t and $\Box_{(l,r)}\neg p$ holds at $t + \epsilon$ for any small ϵ. This implies that p holds somewhere in $t + (l, r)$. Let this point be $t' \in t + (l, r)$. It is possible to pick a sufficiently small ϵ such that $t' > t + l + \epsilon$. But this contradicts the fact that $\boxminus_{(l,r)}\neg p$ holds at $t + \epsilon$.
   Now we show q intervals are of length atleast $r - l$ or are interrupted by BP.
   Let the maximal q interval end at t (0)
   q is false at t as seen above. Since τ' is a canonical extn, we get
   p does not hold anywhere in $t + (l, r)$ (1)
   Case 1: $\neg p$ is true at $t + l$. Then it cannot be the case that $\neg p$ is true in the left neighbourhood of $t + l$ because if it were so, then for some small ϵ, $\neg p$ would hold throughout $t + l - (0, \epsilon]$. This along with (1) and the case assumption we would obtain that p does not hold anywhere in $(t + l - \epsilon, t + r - \epsilon)$ for sufficiently small ϵ. This means q does not hold to the left of t which contradicts (0). Now for any $t' \in (t - r + l, t)$ we have $t + l \in (t' + l, t' + r)$, which implies that for all such t', q holds at t' because p holds to the immediate left of $t + l$ which is contained in $(t' + l, t' + r)$.
   Case 2: p holds at $t + l$. We have already seen that φ_1 holds everywhere, applying at $t + l$ we get the required result.
3. φ_3: this formula says that the q intervals are right open which can be proved in similar fashion as the above.
4. Suppose the LHS of φ_4 is true at t. Then the point $t-r$ exists because of $\neg\diamondminus_{[0,r)}BP$. $\boxminus_{(0,r-l)}\neg p$ implies $\neg\Diamond_{(l,r)}p$ is true at $t - r$. Now along with $\boxminus_{(r,2r-l)}q$ it is easy to see why RHS of φ_4 is true at t.
5. Suppose LHS of φ_5 is true at t. Then we can deduce from the conjuncts containing q that $\neg\Diamond_{(l,r)}p$ holds at $t - r$ and $\Diamond_{(l,r)}p$ holds in its immediate right. If $p \vee \nearrow p$ were not true at t, then using the fact that $\tau', t \models \Box_{(0,r-l)}\neg p$ we can deduce that $\neg p$ should be true throughout $t + (-r + l + \epsilon, \epsilon)$ for a small enough $\epsilon > 0$, which then contradicts the fact that $\Diamond_{(l,r)}p$ holds in the immediate right of $t - r$. Hence φ_5 holds everywhere as well.

Now, we prove the converse. Assume $\tau' \models \boxdot(\varphi_1 \wedge \ldots \varphi_5)$ and show that τ' is canonical extn. (We use the notation $\tau', I \models \varphi$ to say φ holds everywhere in I).

1. If $(\tau', t) \models \Diamond_{(l,r)}p$. Then p holds at $t' \in (t + l, t + r)$. By φ_3, we get that q holds at every point in $(t' - r, t' - l)$ and t lies in this interval, Hence $\tau', t \models q$.
2. If $(\tau', t) \models \neg\Diamond_{(l,r)}p$.
   Consider the earliest point t_1 such that
   $(\tau', [t_1, t]) \models \neg\Diamond_{(l,r)}p$. (1)
   Such a point exists because of two facts
   (a) $\Diamond_{(l,r)}$ intervals are right open(or $\neg\diamondminus_{(l,r)}$ are left closed) and

(b) the time domain under consideration is bounded along the negative axis.
From (1), $(\tau', t_1) \models \neg\Diamond_{(l,r)}p$ which implies
$(\tau, t_1 + r) \models \boxminus_{(0,r-l)}\neg p$ (2)

Now, $(\tau', t_1 + r) \models \boxminus_{(r,2r-l)}q$ (3)
because of the following facts
(a) minimality of t_1
(b) the fact that $\Diamond_{(l,r)}$ intervals are of length atleast $r - l$ or end at BP (A)
(c) for any t, $(\tau', t) \models \Diamond_{(l,r)}p$ implies $\tau', t \models q$ from part 1 (B)
Now using (2) and (3) we apply φ_4 at $t_1 + r$ to obtain
$(\tau', t_1) \models \neg q$ (4)

Now let us assume $(\tau', t) \models q$ (C)
Consider the earliest point $t_2 (< t)$ such that
$(\tau', (t_2, t]) \models q$ and $(\tau', t_2) \models \neg q$ (5)
Such a point exists because q intervals are left open.
Also, $t_2 \geq t_1$ (6)
because of (4) and (5).

Now, $(\tau', t_2 + r) \models \boxminus_{(l,r)}q$ because of (A),(B)
using this with (5) we apply φ_5 at $t_2 + r$ to obtain
$(\tau', t_2 + r) \models p \vee (\nearrow p)$ (7)
Observe that $(\tau', (t_1 + l, t + r)) \models \neg p$ due to (1)
but $t_2 + r \in (t_1 + l, t + r)$ from (5) and (6)
and this contradicts (7).
Hence (C) must be incorrect and $\tau', t \models \neg q$

So τ' is indeed a canonical extn w.r.t. X if it satisfies $\boxdot(\varphi_1 \wedge \ldots \wedge \varphi_5)$. To prove the statement of our theorem: Let $\varphi \in$ BMITL be the given formula. First we replace each $\Diamond_J p$ subformulae in φ with new witness propositions q from P'. Call this formula ψ. Then we add the conjuncts $W_p = (q \Leftrightarrow \Diamond_J p)$ to ψ. We need to remove the conjuncts W_p from ψ. Use $\bigwedge\limits_{p \in P} \nu_p$ to replace this. The resultant formula is $\psi \wedge \bigwedge\limits_{p \in P} \nu_p$. Our final formula ψ_{past} will be $(BP \Rightarrow \psi) \wedge \bigwedge\limits_{p \in P} \nu_p$. □

5 Signal Automata

We define *signal automata* over finite and infinite signals where the the signals can have open/closed/half-open as well as singular intervals and the automaton behaviour can distinguish between these. In this sense, our signal automata are a generalization of signal automata of Maler *et al* [6]. At any time point during its run, a signal automaton can be in in an accepting or non-accepting configuration. The run is “safety accepted” if the automaton stays in accepting configurations throughout the run. Signal automata which accept signals using this safety acceptance criterion are called safety-signal-automata. For finite signals we use acceptance by reaching a final state, and for infinite signals we use Buchi acceptance. Additiona also consider *s*afety-signal automata which use safety-acceptance criterion. We define the important notion of *deterministic* signal automata and we explore the closure properties of signal automata as well as non-emptiness checking.

Given a finite set of clocks $\mathcal{C} = \{x_1, \ldots, x_n\}$, a valuation $\nu \in \mathcal{H}$ with $\mathcal{H} = (\mathbb{R}_{\geq 0})^n$ gives the value of each clock. A configuration of a timed automaton is a pair of the form (q, ν), where q is a state. An atomic clock constraint has the form $x\ op\ c$ where x is a clock, $op \in \{<, \leq, >, \geq\}$ and $c \in \mathbb{N}$.

Definition 1. *A* signal automaton $\mathcal{A}$ *is a tuple* $(Q, Q_0, P, \mathcal{C}, \lambda, Inv, \Delta, F)$ *where (i)* Q *is a finite set of states; (ii)* $Q_0 \subseteq Q$ *is the set of initial states; (iii)* P *is a finite set of propositional variables; (iv)* $\mathcal{C}$ *is a finite set of clocks; (v)* $\lambda : Q \to 2^P$ *associates to every state a set of propositional variables. We call* $\lambda(q)$ *the label of state* q*; (vi) The invariant* Inv *assigns to every state* q *a conjunction of atomic clock constraints of the form* $x \leq m$ *for clock* x*, and* $m \in \mathbb{N}$*; (vii)* Δ *is the transition function. It is a set of tuples* (q, g, u, d, q') *where* g *is a guard,* u *is the update function, and* d *is a bit.* $d = 1$ *specifies that the automaton must be in state* q' *at the time of the transition and* $d = 0$ *specifies that the automaton must be in state* q *at the time of the transition*[2]*. A guard is a boolean combination of atomic clock constraints. An update function* u *specifies for each clock* x *a reset of the form* $x := 0$ *or an assignment of the form* $x := y$ *(to the value of clock* y*); (viii)* $F : Q \to \Phi(C)$ *specifies for each state* q *a boolean combination* $F(q)$ *of atomic clock constraints such that* (q, ν) *is an accepting configuration iff* $\nu \models F(q)$*. clock valuations are given by boolean combinations of clock constraints.*

These automata take finite or infinite signals (as defined in section 2) as input. A run of $\mathcal{A}$ over a signal $\tau = (s_0, I_0)\ (s_1, I_1) \ldots$ is a sequence $(q_0, I'_0, \nu_0) \xrightarrow{\delta_0} (q_1, I'_1, \nu_1) \xrightarrow{\delta_1} (q_2, I'_2, \nu_2) \xrightarrow{\delta_2} \ldots$ where $I'_0 I'_1 I'_2 \ldots$ is an interval sequence (as defined in section 2) which is a refinement of the sequence $I_0 I_1 I_2 \ldots$, with $\lambda(q_j) = s_i$ where $I'_j \subseteq I_i$ for all j. Here, ν_i gives the valuations of the clocks at time I'^{-}_i, the left limit of I'_i. Also $\nu_0 = (0, 0, \ldots, 0)$. Note that the sequence $I'_0 I'_1 I'_2 \ldots$ needs to be a refinment of $I_0 I_1 I_2$ because in the run there is a not only a transition at then end of every I_i but there may be a transition at a point inside I_i. A step $(q, I, \nu) \xrightarrow{\delta} (q', I', \nu')$ where δ is a transition (q, g, u, d, q') is taken at I^+, the right limit of I, provided
(1) If $d = 1$, then I' must be left-closed and I is right-open, and if $d = 0$, then I must be right-closed and I' is left-open.
(2) $\forall t \in I, \nu + (t - I^-) \models Inv(q)$, and $\nu' = u(\nu + I^+)$,
(3) $(\nu + I^+) \vDash g$, and $\forall t' \in I', \nu' + (t' - I'^-) \models Inv(q')$. For the above run ρ, the set $Accset(\rho)$ gives the set of time points when the automaton is in an accepting configuration, i.e. for any $t \in I'_i$ we have $t \in Accset(\rho)$ iff $\nu_i + t - I'^-_i \models F(q_i)$.

Acceptance. A run ρ accepts time point $t \in Dom(\tau)$ if it is in accepting configuration at time t, i.e. $t \in Accset(\rho)$. The run is called safety-accepting if it accepts all $t \in$

[2] Pictorially, a transition with $d = 1$ is denoted by $(q, I, \nu) \xrightarrow{\delta\ \bullet} (q', I', \nu')$ and one with $d = 0$ is denoted as $(q, I, \nu) \xrightarrow{\bullet\ \delta} (q', I', \nu')$. Whenever the $\bullet$ omitted, it means a pair of transitions, one of the form $\xrightarrow{\delta\ \bullet}$ and the other of the form $\xrightarrow{\bullet\ \delta}$.

$Dom(\tau)$. The automaton safety-accepts the signal τ if it has a run on τ which is safety accepting. The resulting automaton is called a *safety-signal-automaton* and denoted by $\boxdot A$. Note that safety acceptance is identically defined for signal automata over both finite and infinite signals. A signal-safety-automaton is called *unambiguous* if for any signal τ it has at most one accepting run.

Determinism. A signal automaton is said to be *deterministic* if it satisfies:

(I) For any two initial states $q_0, q_0' \in Q_0$, $\lambda(q_0) \neq \lambda(q_0')$.
(A) For two distinct transitions $(q, g_1, \rho_1, d_1, q_1)$ and $(q, g_2, \rho_2, d_2, q_2)$,
  (A1) $\lambda(q_1) \neq \lambda(q_2)$ or
  (A2) If $\lambda(q_1) = \lambda(q_2) \neq \lambda(q)$, then either $Inv(q) \Rightarrow \neg(g_1 \wedge g_2)$, or $d_1 \neq d_2$.
  (A3) If $\lambda(q_1) = \lambda(q_2) = \lambda(q)$, then $Inv(q) \Rightarrow \neg(g_1 \wedge g_2)$.
(B) For every transition (q, g, ρ, d, q')
  (B1) $\lambda(q) \neq \lambda(q')$ or
  (B2) If $\lambda(q) = \lambda(q')$, then $Int(Inv(q)) \cap g = \emptyset$ and $d = 0$.

The following lemmas give some useful properties of deterministic signal automata. Their proof may be found in the full verison of the paper.

Lemma 5. *Let $\mathcal{A}$ be a deterministic signal automaton and τ be a signal. Then $\mathcal{A}$ has a unique run on τ.* □

Lemma 6. *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are two deterministic signal automata, then the product $\mathcal{A}_1 \times \mathcal{A}_2$ is also deterministic.*

Boolean closure Deterministic signal automata are closed under boolean operations in the following sense. Let $Accset(\mathcal{A}, \tau)$ denote $Accset(\rho)$ where ρ is the unique run of $\mathcal{A}$ on τ.

Lemma 7. *Given deterministic signal automata A_1 and A_2, we can construct deterministic signal automata $(\neg A_1)$, $A_1 \cap A_2$ and $A_1 \cup A_2$ such that for any signal τ, $Accset(\neg A_1, \tau) = Dom(\tau) - Accset(A_1, \tau)$, $Accset(A_1 \cap A_2, \tau) = Accset(A_1, \tau) \cap Accset(A_2, \tau)$, and $Accset(A_1 \cup A_2, \tau) = Accset(A_1, \tau) \cup Accset(A_2, \tau)$.*

Definition 2. *Let $\boxdot\mathcal{A}$ be a safety-signal-automaton accepting signals over a set of propositions P'. Let $P \subseteq P'$. The safety-signal-automaton $\boxdot\mathcal{A} \restriction P$ is constructed by projecting $\mathcal{A}$ onto P. This is obtained by restricting labelling function λ in $\mathcal{A}$ to P (or dropping propositions in $\lambda(q)$ not in P). Note that size of $\boxdot\mathcal{A} \restriction P$ is same as the size of $\boxdot\mathcal{A}$ but it may be non-deterministic even if $\mathcal{A}$ is deterministic.*

Lemma 8. *Let $\mathcal{A}$ be an automaton accepting signals over a set of propositions P'. Let $P \subseteq P'$. Then $L(\mathcal{A}) \restriction P = L(\mathcal{A} \restriction P)$.* □

6 MITL to Signal Automata

In Section 4, we have given a reduction from $\mathsf{MITL}_{\mathsf{fin}}[\mathcal{U}, \diamond_I, \mathcal{S}, \diamondminus_I]$ (or BMITL) formula ϕ to language equivalent formula $\exists P \boxdot \psi$ where ψ is a pure past-time formula of MITL. Maler *et al* [6] gave a synthesis of deterministic signal automaton (with end-state based acceptance) for finite signals and this was extended to infinite signals in [7]. In this section, we adapt this result to the more general setting of all finitely variable signals. Making use of this deterministic signal automaton for the past time formula ψ, we obtain a straightforward construction of (nondeterministic) safety-signal-automaton for $\exists P \boxdot \psi$.

Lemma 9. *Given a formula φ in $\mathsf{MITL}_{\mathsf{fin}}[\mathcal{S}, \diamondminus_I]$ or $\mathsf{MITL}_{\mathsf{inf}}[\mathcal{S}, \diamondminus_I]$, we can construct a deterministic signal automaton $\mathcal{A}_\varphi$ such that*

- *for any signal τ and $t \in Dom(\tau)$ we have $\tau, t \models \varphi$ iff $t \in Accset(\mathcal{A}_\varphi, \tau)$.*
- *Hence, $L(\boxdot\mathcal{A}_\varphi) = \{\tau \mid \tau \vDash \boxdot\varphi\}$.*

Proof (sketch). The construction is a variant of the construction of Maler *et al* [6]. We start with the assumption that formula ϕ in $\mathsf{MITL}_{\mathsf{fin}}[\mathcal{S}, \diamondminus_I]$ (or $\mathsf{MITL}_{\mathsf{inf}}[\mathcal{S}, \diamondminus_I]$) is already flattened. For each conjunct involving one temporal operator, we construct a deterministic signal automaton. Their intersection automaton (as given by Lemma 7) gives the desired deterministic signal automaton for ϕ. In the full version of the paper, we show the construction of deterministic automata corresponding to the operators $\diamondminus_{(a,b)}p$, $\diamondminus_{(a,\infty)}p$ and $p\,\mathcal{S}q$. Below we only explain the construction of the automaton for the formula $\diamondminus_{(a,\infty)}p$.

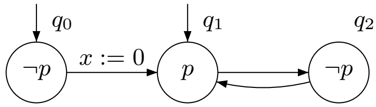


Fig. 1. Automata for $\diamondminus_{(a,\infty)}p$

Define $\mathcal{C} = (Q, \{q_0, q_1, q_2\}, \{p\}, \{x\}, \lambda, Inv, \Delta, F)$, where Δ, λ are as depicted in Figure 1. $F = \{(q, x > a) \mid q \in \{q_1, q_2\}\}$ and Inv is $true$ for all states. It is easy to see that the signal automaton is deterministic since there are no transitions between states of the same label. p may remain false initially; when p becomes true, the clock is reset. The accepting configurations $(q_1, x > a)$ and $(q_2, x > a)$ ensure that a p was seen in (a, ∞) in the past. □

Theorem 4. *For any formula $\varphi \in \mathsf{MITL}_{\mathsf{fin}}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ (or BMITL), we can construct an unambiguous safety signal automaton $\boxdot\mathcal{A}$ such that $L(\boxdot\mathcal{A}) = \{\tau \mid \tau, 0 \models \varphi\}$.*

Proof. Starting from $\varphi \in \mathsf{MITL}_{\mathsf{fin}}[\mathcal{U}, \Diamond_I, \mathcal{S}, \diamondminus_I]$ (or BMITL) over P, we will obtain (using Theorems 2, 3) a formula $\varphi' \in \mathsf{MITL}_{\mathsf{fin}}[\mathcal{S}, \diamondminus_I]$ (or $\mathsf{MITL}_{\mathsf{inf}}[\mathcal{S}, \diamondminus_I]$) over $P \cup P'$ such that $L(\varphi) = L(\exists P' \boxdot \varphi')$. Note that the size of this formula is polynomial in size of the original formula. As observed in Lemma 9, we can construct a deterministic safety signal automaton $\boxdot\mathcal{A}$ accepting $L(\boxdot\varphi')$. Now consider the automaton $\boxdot\mathcal{A} \upharpoonright P$ obtained by dropping from the states of $\boxdot\mathcal{A}$, the propositions in P', as in Lemma 8. Clearly, $L(\boxdot\mathcal{A} \upharpoonright P) = L(\exists P' \boxdot \varphi')$. We argue that $\mathcal{A} \upharpoonright P$ is unambiguous.

Assume to contrary that τ is a signal which has two distinct accepting runs ρ_1 and ρ_2 in $\boxdot\mathcal{A} \upharpoonright P$. Then, $\tau \models \exists P' \boxdot \phi$. As shown in proofs of Theorems 2 and 3 there exists a canonical extension τ' such that $\tau' \models \boxdot\varphi'$. Hence, the automaton $\boxdot\mathcal{A}$ has the two distinct accepting runs on τ'. But as $\boxdot\mathcal{A}$ is a determinstic safety signal automaton, this is impossible, giving a contradiction. □

6.1 Complexity

We now investigate the size of the signal automaton that we have constructed for a MITL formula φ. Each base formula $\diamondminus_{(a,b)}p$ requires $\mathcal{O}(\frac{b}{b-a})$ states and clocks [6]. As a and b are integers it becomes $\mathcal{O}(b)$. Let B denote the max b appearing in any interval used in ϕ. Then the automaton we construct will have $\mathcal{O}(B^{|\phi|})$ states and $\mathcal{O}(B|\phi|)$ clocks with maximum clock constant B. The size of the automaton for $\exists P' \boxdot \varphi$ is same as the size of the automaton for φ, except that it can be non-deterministic. The constructed automaton can be used to check satisfiability of φ using standard region construction: this results in an EXPSPACE algorithm for satisfiability checking.

References

1. Alur, R., Henzinger, T.A.: Logics and Models of Real Time: A Survey. In: Proceedings of REX Workshop, pp. 74-106 (1991)
2. Alur, R., Feder, T., Henzinger, T.A.: The Benefits of Relaxing Punctuality. Journal of the ACM 43(1), 116–146 (1996)
3. D'Souza, D., Raj Mohan, M., Prabhakar, P.: Eliminating past operators in Metric Temporal Logic. In: Perspectives in Concurrency, pp. 86–106. Universities Press (2008)
4. Henzinger, T.A.: The Temporal Specification and Verification of Real-time Systems. Ph.D Thesis, Stanford Unuiversity (1991)
5. Koymans, R.: Specifying Real-Time Properties with Metric Temporal Logic. Real Time Systems 2(4), 255–299 (1990)
6. Maler, O., Nickovic, D., Pnueli, A.: Real Time Temporal Logic: Past, Present, Future. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 2–16. Springer, Heidelberg (2005)
7. Maler, O., Nickovic, D., Pnueli, A.: On Synthesizing Controllers from Bounded-Response Properties. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 95–107. Springer, Heidelberg (2007)
8. Nickovic, D., Piterman, N.: From MTL to Deterministic Timed Automata. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 152–167. Springer, Heidelberg (2010)
9. Ouaknine, J., Worrell, J.: On the Decidability of Metric Temporal Logic. In: Proceedings of LICS 2005, pp. 188–197 (2005)
10. Ouaknine, J., Worrell, J.: On Metric Temporal Logic and Faulty Turing Machines. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 217–230. Springer, Heidelberg (2006)
11. Prabhakar, P., D'Souza, D.: On the Expressiveness of MTL with Past Operators. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 322–336. Springer, Heidelberg (2006)

Craig Interpolation in the Presence of Non-linear Constraints*

Stefan Kupferschmid and Bernd Becker

Albert-Ludwigs-Universität Freiburg, Germany
{skupfers,becker}@informatik.uni-freiburg.de

Abstract. An increasing number of applications in particular in the verification area leverages Craig interpolation. Craig interpolants (CIs) can be computed for many different theories such as: propositional logic, linear inequalities over the reals, and the combination of the preceding theories with uninterpreted function symbols. To the best of our knowledge all previous tools that provide CIs are addressing decidable theories. With this paper we make Craig interpolation available for an in general undecidable theory that contains Boolean combinations of linear and non-linear constraints including transcendental functions like $\sin(\cdot)$ and $\cos(\cdot)$. Such formulae arise e.g. during the verification of hybrid systems. We show how the construction rules for CIs can be extended to handle non-linear constraints. To do so, an existing SMT solver based on a close integration of SAT and Interval Constraint Propagation is enhanced to construct CIs on the basis of proof trees. We provide first experimental results demonstrating the usefulness of our approach: With the help of Craig interpolation we succeed in proving safety in cases where the basic solver could not provide a complete answer. Furthermore, we point out the (heuristic) decisions we made to obtain suitable CIs and discuss further possibilities to increase the flexibility of the CI construction.

Keywords: SAT, SMT, Craig Interpolation, Interval Arithmetic, BMC.

1 Introduction

The analysis and verification of hybrid systems is an important task, e.g. in the automotive or aviation industry. Wherever complex systems are developed for applications with safety critical aspects, the developers must fulfill a number of safety requirements, which in general slows down the process of development and increases its costs. This has motivated the development of tools that can deal with the verification task of such systems. One technique widely used in multiple verification tools is Craig interpolation. In this paper we present a method that allows the construction of *Craig interpolants* (CIs) [1] for arbitrary Boolean combinations of linear and non-linear constraint formulae.

* This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). Refer to www.avacs.org.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 240–255, 2011.

CIs are of particular use in bounded model checking (BMC). In [2] McMillan extended the traditional BMC procedure for Kripke structures to an unbounded model checking algorithm, i.e. a procedure that is able to show that a given system is safe in the sense that a certain safety property always holds. In [3,4] this work is extended to the quantifier-free theory of linear inequalities and uninterpreted function symbols. The authors of [5] compute optimized representations for non-convex polyhedra with the help of Craig interpolation. More in detail, Craig interpolation here is used to remove redundant linear constraints with the aid of a *Satisfiability Modulo Theories* (SMT) solver. A representative for a state-of-the-art SMT solver that can produce CIs is presented in [6], but this work is limited to linear constraints. Since we focus on verification of hybrid systems, we are interested in a solver that can deal with the inherent linear and non-linear behavior of such systems. Our contributions are: (i) we generalize proof-based construction rules for CIs [2] to formulae containing non-linear constraints and prove the correctness of the rules, (ii) we built these modifications on top of the SMT solver iSAT [7], (iii) experimental results illustrate that the so computed CIs can be used in a similar fashion as suggested by [2] to verify safety properties but this time for systems containing non-linear dynamics; last but not least, we discuss the choices we made for the CI construction and discuss further possibilities to influence the construction of CIs and possibly find "good" CIs.

The remainder of this paper is structured as follows: First we describe the core of iSAT, the underlying solver used in this paper and then show how this solver can produce proofs of unsatisfiability. Next we present the construction rules to achieve valid CIs. We then present some promising results, illustrating that interpolation can be successfully applied to BMC problems containing non-linear constraints. Before concluding the paper we discuss how the strength of a CI can be influenced by analysing the slackness between contradictionary theory constraints that are detected during the solving process of iSAT.

2 Foundations

As already mentioned above, on the one hand side we of course need a solver that is able to handle linear as well as non-linear constraints, and on the other hand provides a proof in the case of unsatisfiability. In order to construct CIs and do the experimental work, we modified the SMT solver iSAT [7] in a corresponding way. Nevertheless, the main construction principles together with the applications presented should be transferable to any SMT solver that is capable of handling linear and non-linear constraints.

Since the solver underlying our work is iSAT, we provide a short description of the solver as far as it is necessary for the understanding on generating CIs. For more information on iSAT refer to [7].

2.1 Basics on iSAT

The iSAT algorithm aims at solving Boolean combinations of mixed linear and non-linear constraint formulae (including transcendental functions). The undecidability of this theory in general follows from the fact that it is undecidable to answer whether a Diophantine equation has an integer solution or not [8].

iSAT contains an integration of a Davis-Putnam-Logemann-Loveland (DPLL) procedure [9,10] and interval constraint propagation (ICP)[1] allowing it to reason about (highly non-linear) arithmetic constraints. In contrast to decidable theories, iSAT can not always classify a problem as "satisfiable" or "unsatisfiable". Instead, the result can also be "unknown". However, iSAT can handle arithmetic expressions like $\sin(\cdot)$, $\cos(\cdot)$ or $\exp(\cdot)$ which are not supported by common SMT solvers.

The solving process of iSAT consists of two phases. First iSAT transforms an arbitrary Boolean combination of linear and non-linear constraints into an equisatisfiable *Conjunctive Normal Form* (CNF) with normalized constraints using the following syntax:

$$\begin{aligned}
\mathit{formula} &::= \{\mathit{clause} \wedge\}^* \mathit{clause} \\
\mathit{clause} &::= (\{\mathit{atom} \vee\}^* \mathit{atom}) \\
\mathit{atom} &::= \mathit{simple\_bound} \mid \mathit{arithmetic\_predicate} \\
\mathit{simple\_bound} &::= \mathit{variable}\ \mathit{relation}\ \mathit{rational\_const} \\
\mathit{arithmetic\_predicate} &::= \mathit{variable}\ \mathit{relation}\ \mathit{uop}\ \mathit{variable} \mid \\
&\ \mathit{variable}\ \mathit{relation}\ \mathit{variable}\ \mathit{bop}\ \mathit{variable} \\
&\ \mathit{variable}\ \mathit{relation}\ \mathit{rational\_const}\ \mathit{bop}\ \mathit{variable}
\end{aligned}$$

In the above syntax, *uop* and *bop* are unary and binary operation symbols respectively, including $+$, $-$, $\times$, $\sin(\cdot)$, etc., *rational_const* ranges over the rational constants, and $\mathit{relation} \in \{<, \leq, =, \geq, >\}$. To illustrate this phase, imagine that we have the following formula:

$$(x \geq 0) \wedge (x \leq 10) \wedge ((\sin(1/3x) + \sqrt{x} \geq y) \implies (y \geq 1/4x + 3)) \tag{1}$$

First we eliminate the Boolean operators by applying a Tseitin-transformation [12], e.g. the implication will be replaced by a new auxiliary Boolean variable (b). The remaining formula is then normalized by introducing additional *real* variables r_1, r_2 and r_3 and the following constraints $r_1 = 1/3x$, $r_2 = \sin(r_1)$ and $r_3 = \sqrt{x}$. Finally, the normalized CNF problem looks like follows:

$$\begin{aligned}
&(b) \wedge (x \geq 0) \wedge (x \leq 10) \wedge (\overline{b} \vee r_2 + r_3 < y \vee y \geq r_4 + 3) \wedge \\
&(r_2 + r_3 \geq y \vee b) \wedge (y < r_4 + 3 \vee b) \wedge \\
&(r_1 = 1/3x) \wedge (r_2 = \sin(r_1)) \wedge (r_3 = \sqrt{x}) \wedge (r_4 = 1/4x)
\end{aligned} \tag{2}$$

Now all clauses are consistent with the syntax described above and can be transferred to the solver. In the remainder of the paper we will assume that a normalization has been performed in advance and thus the formula φ considered is normalized.

Before describing the solving process in detail, we informally define the underlying semantics. A constraint formula φ is satisfied by a valuation of its variables

[1] cf. [11] for an extensive survey.

iff all its clauses are satisfied, that is, iff at least one atom is satisfied in any clause. An atom is satisfied wrt. the standard interpretation of the arithmetic operators and the ordering relations over the reals. A constraint formula φ is *satisfiable* iff there exists a satisfying valuation, referred to as a *solution* of φ. Otherwise, φ is *unsatisfiable*. We remark that by definition of satisfiability, a formula φ including or implying the empty clause, denoted by $\bot$, cannot be satisfied at all, i.e. if $\bot \in \varphi$ or $\varphi \implies \bot$ then φ is unsatisfiable.

Instead of real-valued variable valuations, iSAT manipulates interval ranges. Using the function $\rho : \mathit{Var} \to \mathbb{I}_{\mathbb{R}}$, where Var is a set of variables and $\mathbb{I}_{\mathbb{R}}$ is the set of convex subsets of $\mathbb{R}$, we define a range for each variable. Note, that we also support discrete variable domains (integer and Boolean). To this end, it suffices to clip the interval of integer variables accordingly, such that $[-3.4, 6.0)$ becomes $[-3, 5] \subset \mathbb{Z}$, for example. The Boolean domain is represented by $\mathbb{B} = [0, 1] \subset \mathbb{Z}$. If both ρ' and ρ are interval valuations, then ρ' is called a *refinement* of ρ iff $\rho'(v) \subseteq \rho(v)$ for each variable $v \in \mathit{Var}$. The lower and upper interval borders of an interval $\rho(x)$ for a variable x can be encoded as simple bounds. We denote the lower and upper interval border of the interval $\rho(x)$ by $\mathit{lower}(\rho(x))$ and $\mathit{upper}(\rho(x))$, respectively. E.g., for the interval $\rho(x) = (-4, 9]$ we have $\mathit{lower}(\rho(x)) = (x > -4)$ and $\mathit{upper}(\rho(x)) = (x \leq 9)$.

Let x and y be variables, ρ be an interval valuation, and $\circ$ be a binary operation. Then $\rho(x \circ y)$ denotes the *interval hull* of $\rho(x) \hat{\circ} \rho(y)$ (i.e. the smallest enclosing interval which is representable by machine arithmetic), where the operator $\hat{\circ}$ corresponds to $\circ$ but is canonically lifted to sets. This is done analogously for unary operators. We say that an atom a is *inconsistent* under an interval valuation ρ, referred to as $\rho \sharp a$, iff no values in the intervals $\rho(x)$ of the variables x in a satisfy the atom a, i.e.

$$\begin{array}{lll} \neg\exists v \in \rho(x) & : v \sim c & \text{if } \; a = (x \sim c), \\ \neg\exists v \in \rho(x), \neg\exists v' \in \rho(\circ y) & : v \sim v' & \text{if } \; a = (x \sim \circ y), \\ \neg\exists v \in \rho(x), \neg\exists v' \in \rho(y \circ z) & : v \sim v' & \text{if } \; a = (x \sim y \circ z) \end{array}$$

where $\sim \in \{<, \leq, =, \geq, >\}$. Otherwise a is *consistent* under ρ. For instance, the constraint $x = 3 \cdot y$ is inconsistent for $\rho(x) = [5, 10]$ and $\rho(y) = [-2, 1]$.

For our purpose we do not need the definition of interval satisfaction. It is sufficient to talk about atoms which are still consistent. We remark that proving the satisfiability of an iSAT formula is not trivial. For more details confer [7]. In Algorithm 1, the pseudocode of iSAT is given. Before the main iSAT routine starts, it is assumed that all the unit clause information contained in the original formula has already been propagated, which can sometimes allow us to derive tighter bounds. Once this is ensured, Algorithm 1 begins by making a decision, and splitting the interval range of a variable, e.g. splits a variable's range in half (line 3). This decision will be propagated in line 4. If a conflict is detected (e.g. an evaluated clause becomes inconsistent during propagation) it will be analyzed in line 5. The conflict analysis routine uses the implication graph of the solver to compute the reasons for the conflict. By doing so a conflict clause is learned, allowing iSAT to prune off unsatisfiable parts of the search space. iSAT terminates in either line 5 or 8 with either *unsat*, *sat*, or *unknown*.

```
Data: CNF F
Result: sat, unsat or unknown
/* Main DPLL loop. DecideVar returns false once the msw for all      */
/* variables is reached, and no further decisions are possible.     */
while decideVar() do
    /* Propagates current decision and unit constraints.            */
    if propagateICP() = $Conflict$ then
        /* Function tries resolve the conflict by backtracking. If  */
        /* the conflict is unresolvable, problem is unsatisfiable.  */
        if analyseBacktrack() = $Unresolvable$ then return $unsat$;
    end
end
/* Final test to see if all the constraints are satisfied.          */
if allClausesSat() then return $sat$; else return $unknown$;
```

Algorithm 1. DPLL + ICP

As termination can not be guaranteed by dividing an interval range indefinitely, iSAT stops making decisions when every problem variable has reached a current interval width that is less or equal to a given minimal width. We call this width the minimal splitting width (*msw*). It is still possible for this current ρ to contain a solution, as such, this result is sometimes referred to as a *Candidate Solution*. However, if iSAT found an arising conflict as unresolvable during the conflict analysis routine the problem formula is classified as unsatisfiable. As our construction of CIs is proof-based, we describe in the next section how these certificates can be computed.

2.2 Proof Certificates in iSAT

This section summarises the work of [13] where iSAT is used to produce certificates. We introduce the two rules used in iSAT and provide simple examples. For a detailed description refer to [13].

The first rule is based on unit propagation, and the second one is based on resolution. Conceptually, the rules from modern SAT solvers for proof generation are adapted to our context.

In order to apply the deduction rule, there must be a clause *cl* that contains at most one atom a_i that does not evaluate to $false$ under the current interval assignment ρ. In other words, clause *cl* is either a unit-clause with its unit-literal a_i, or it is the conflicting clause. In order to extend ρ to a satisfying assignment we have to satisfy atom a_i. With the help of interval arithmetic, iSAT tries to derive new upper and/or lower bounds for the variables contained in a_i. Assume we want to solve a CNF formula φ that contains the clause $cl = (x < -8 \vee y = x^2)$, and the current variable intervals defined by ρ are $x \in [3, 7]$ and $y \in [-2, 25]$. Here, the first atom of *cl* is inconsistent because there exists no value between 3 and 7 that is less than -8 $(\rho \sharp (x < -8))$. The second atom, $y = x^2$, is therefore the unit literal of *cl*. The reason why a clause *cl* is a unit-clause is given by a subset of the current interval assignment ρ containing all current lower and upper bounds that are responsible for the inconsistency of the remainder atoms ($reason\_unit(cl, y = x^2, \rho) = \{x \geq 3\}$). Now, we know that $x \in [3, 7]$ and we further have the unit literal $y = x^2$. Putting this information together we see that y must be in the interval $[9, 49]$. Since the current range is $y \in [-2, 25]$, we can derive a new lower bound for variable y that is $y \geq 9$ by applying ICP $((x \geq 3) \overset{y=x^2}{\rightsquigarrow} (y \geq 9))$, and prune away unsatisfiable parts of the search space. Differently spoken, the clause derived by deduction contains the negation of the reasons and the derived new bound information as literals. In the more general case, we write $c_1 \rhd c_2$ to express that c_2 can be derived by applying the deduction rule on c_1. More formally, this rule is defined as follows:

$$\frac{\begin{array}{c} cl = (a_1 \vee \ldots \vee a_n), \\ \exists \rho : \exists i \in \{1, \ldots, n\} : \forall j \neq i : \rho \sharp a_j, (b'_1, \ldots, b'_k) \overset{a_i}{\rightsquigarrow} (b')\,, \\ \{b'_1, \ldots, b'_k\} \subseteq \{lower(\rho(x)), upper(\rho(x)) : x \in a_i\}, \\ reason\_unit(cl, a_i, \rho) = \{b_1, \ldots, b_m\} \end{array}}{(\neg b_1 \vee \ldots \vee \neg b_m \vee \neg b'_1 \vee \ldots \vee \neg b'_k \vee b')} \tag{3}$$

Therefore the deduction rule for the just presented example is:

$$\frac{\begin{array}{c} cl = (x < -8 \vee y = x^2), \\ \rho \sharp (x < -8), (x \geq 3) \overset{y=x^2}{\rightsquigarrow} (y \geq 9)\,, \\ reason\_unit(cl, y = x^2, \rho) = \{x \geq 3\} \end{array}}{(x < 3 \vee y \geq 9)}$$

The second rule used in iSAT is resolution. In state-of-the-art SAT solvers, resolution is performed during conflict analysis. The same holds for iSAT. This rule can be defined as:

$$\frac{\begin{array}{c} c_1 = (a \vee a_1 \vee \ldots \vee a_n), c_2 = (b \vee b_1 \vee \ldots \vee b_m) \\ a, a_1, \ldots, a_n, b, b_1, \ldots, b_m \text{ are simple bounds} \\ a = (x \sim k), b = (x \sim' k'), \{v : v \sim k\} \cap \{v : v \sim' k'\} = \emptyset \end{array}}{c_{res} = (a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m)} \tag{4}$$

Resolution is applied on two clauses c_1 and c_2 where both clauses contain only simple bounds. In order to apply this rule, c_1 must contain a simple bound a that is contradictory to a simple bound b from clause c_2. To illustrate this rule consider the following two clauses: $c_1 = (x > 4 \vee y \leq 6 \vee z < 5)$ and $c_2 = (x < -8 \vee w > 3)$. The simple bound $x > 4$ from clause c_1 and the simple bound $x < -8$ from clause c_2 cannot be valid at the same time. By resolving these two simple bounds we generate the following resolvent $c_{res} = (y \leq 6 \vee z < 5 \vee w > 3)$. We just write $res(c_1, c_2, p) \vdash c_{res}$ as an abbreviation for the fact that c_{res} can be derived by applying the resolution rule to clauses c_1 and c_2 on variable p. Before presenting the construction of CIs in iSAT in the next section, we want to take a deeper look at the resolution steps. At first glance the resolution rule is very similar compared to resolution between two propositional clauses. But in the just presented example the resolved simple bounds $x > 4$ and $x < -8$ contain besides the contradictory information also information about the *strength* of the contradiction. We call this *slackness information* and it can be obtained by computing the distance between these bounds, here $|4 - (-8)| = 12$. This is one of the main reasons why we implemented the generation of CIs in an interval arithmetic based SMT solver as we want to use the slackness information in later versions of iSAT to influence the construction of these interpolants. In order to understand the construction of CIs in iSAT, it is important to know the two rules, namely resolution and deduction. To illustrate how iSAT uses these rules, we present a small example. Suppose iSAT has to solve the normalized CNF φ:

$$\varphi = c_1 \wedge \overbrace{(\sin(x) < 0.3 \vee y < 7.5)}^{c_2} \wedge \overbrace{(y - x \leq 8 \vee b)}^{c_3} \wedge c_4 \wedge \ldots \wedge c_n$$

Here x and y are real valued variables and b is a Boolean variable. At each step iSAT maintains the current interval valuation ρ. Suppose that the current interval valuation is: $x \in [0, 1]$, $y \in [7, 10]$, and b is unassigned. Next, iSAT makes a decision on variable y by splitting the current interval at its midpoint, e.g. $y \in [8.5, 10]$. While propogating this decision, iSAT detects that clause c_2 is now unit as literal $(y < 7.5)$ evaluates to *false* under the current interval valuation. The unit-literal is $\sin(x) < 0.3$ and is used to fuel deduction. iSAT then derives a new upper bound $x \leq 0.31$ for variable x (more in detail, based on the deduction rule we may conclude $c_2 \rhd (y < 8.5 \vee x > 1 \vee x \leq 0.31)$). Again no conflicts are encountered, so iSAT is free to make another decision, and for instance sets b to *false*. With this decision clause c_3 becomes unit and literal $(y - x \leq 8)$ is used for deduction. Under the current interval valuation of x and y a new upper bound for variable y via ICP is deduced: $y \leq 8.31$ $(c_3 \rhd (b \vee x > 0.31 \vee y \leq 8.31))$. This newly derived bound conflicts with the current interval valuation $y \in [8.5, 10]$. iSAT resolves this conflict by analyzing the implication graph. Afterwards a conflict clause is learned to prune away unsatisfiable parts of the search space. During the conflict analysis a partial proof will be generated based on the two rules namely deduction and resolution. This partial proof is shown in Fig. 1. Moreover, iSAT has been modified to produce so-called partial CIs during conflict analysis.

As usual, we define $sup(F)$ to be the set of variables contained in a formula F. Then, according to [1] a CI is defined as follows.

Definition 1 (Craig Interpolant (CI)). *Let A and B be formulae with the property that $A \wedge B$ is unsatisfiable. A formula I is referred to as a CI if the following three properties hold:*

1. $sup(I) \subseteq sup(A) \cap sup(B)$
2. $\models A \Rightarrow I$
3. $\models I \Rightarrow \neg B$

We call I a *Craig interpolant* or just *interpolant* for the formula pair A and B. This interpolant contains only variables occurring in $sup(A) \cap sup(B)$ and can be seen as an over-approximation of the formula A that is completely disjoint

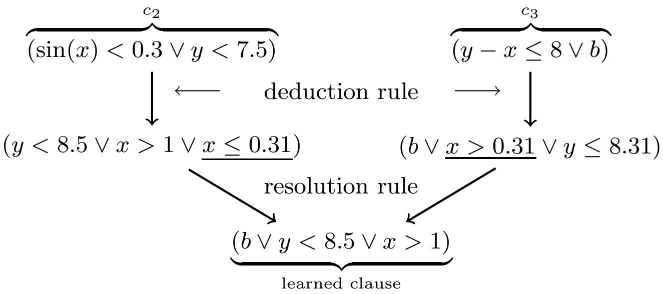

$\overbrace{(\sin(x) < 0.3 \vee y < 7.5)}^{c_2}$
$\overbrace{(y - x \leq 8 \vee b)}^{c_3}$
deduction rule
$(y < 8.5 \vee x > 1 \vee \underline{x \leq 0.31})$
$(b \vee \underline{x > 0.31} \vee y \leq 8.31)$
resolution rule
$\underbrace{(b \vee y < 8.5 \vee x > 1)}_{\text{learned clause}}$

Fig. 1. Partial Proof Tree

from B (i.e. the intersection of I and B is empty). Now, using these basics, the next sections will show how CIs are calculated and how they are used in hybrid system verification by iSAT.

3 Construction of Craig Interpolants Using iSAT

In Fig. 1 of Sec. 2 we showed how iSAT produces a partial proof. In this section we present how iSAT can compute *partial interpolants* using a partial proof. First we provide the construction rules, before proving their soundness. Furthermore, we will apply these rules to the partial proof shown in Fig. 1.

With every formula pair (A, B) we associate the following three sets of variables: 1) G contains all the variables occurring in both formulae, i.e. $G = sup(A) \cap sup(B)$, 2) L_A contains only variables located in the A-formula but not in the B-formula, and 3) L_B and contains all variables from the B-formula but not located in the A-formula. We assume that the formula pair (A, B) is in CNF format and we have a proof tree that derives the empty clause from $A \wedge B$. For every node c in the proof tree (internal nodes correspond to clauses derived by applying deduction or resolution) we define construction rules that generate partial interpolants. Note, all internal nodes of the proof tree, including the empty clause $\perp$, contain only simple bounds. With this in mind, we define the construction rules and the concept of projection as follows:

Definition 2 (Construction Rules). *Let c be a node in the proof tree corresponding to a clause containing only simple bounds. Then we denote the partial interpolant of clause c by $pi(c)$, and define*

$$pi(c) = \begin{cases} \bigvee_{l_j \in c, var(l_j) \in G} l_j & : \quad (c \in A) \ or \ (\exists c' \in A : c' \rhd c) \\ pi(c_1) \wedge pi(c_2) & : \quad res(c_1, c_2, p) \vdash c \ and \ p \in G \cup L_B \\ pi(c_1) \vee pi(c_2) & : \quad res(c_1, c_2, p) \vdash c \ and \ p \in L_A \\ true & : \quad else \end{cases}$$

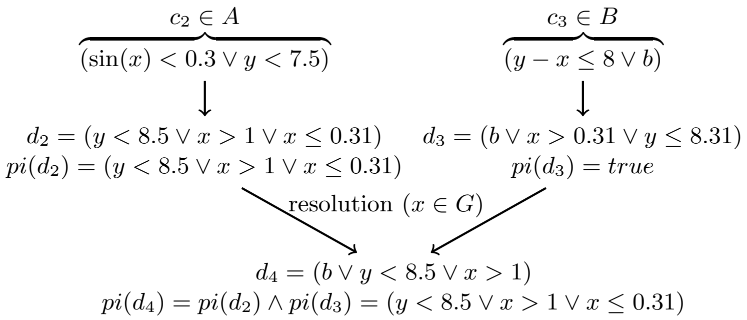

$c_2 \in A$
$(\sin(x) < 0.3 \vee y < 7.5)$
$c_3 \in B$
$(y - x \leq 8 \vee b)$
$d_2 = (y < 8.5 \vee x > 1 \vee x \leq 0.31)$
$pi(d_2) = (y < 8.5 \vee x > 1 \vee x \leq 0.31)$
$d_3 = (b \vee x > 0.31 \vee y \leq 8.31)$
$pi(d_3) = true$
resolution $(x \in G)$
$d_4 = (b \vee y < 8.5 \vee x > 1)$
$pi(d_4) = pi(d_2) \wedge pi(d_3) = (y < 8.5 \vee x > 1 \vee x \leq 0.31)$

Fig. 2. Partial Interpolant

Definition 3 (Projection). *Let Θ be a disjunction of simple bound literals $\Theta = \bigvee_{j=1}^{k} l_j$ with $var(l_j) \in G \cup L_A \cup L_B$. Then, $\Theta \mid_A$ is the projection of Θ to L_A and $\Theta \mid_B$ is the projection of Θ to $L_B \cup G$:*

$$\Theta \mid_A = \bigvee_{var(l_j) \in L_A}^{k} l_j \qquad\qquad \Theta \mid_B = \bigvee_{var(l_j) \in G \cup L_B}^{k} l_j$$

In order to prove the correctness of the construction rules we prove the soundness of the following lemma:

Lemma 4. *Let $F = A \wedge B$ be a CNF formula that is unsatisfiable and let P be a proof of the unsatisfiability. Then for every internal proof node c, the partial interpolant $pi(c)$ is a CI of the formula pair (A', B') with $A' = A \wedge \neg(c \mid_A)$ and $B' = B \wedge \neg(c \mid_B)$.*

Before giving a proof sketch of the lemma above we note the following: Lemma 4 implies that $pi(\bot)$ is a valid CI for the formula pair (A, B). This is clear as $A' = A \wedge \neg(\bot|_A) = A \wedge true = A$ and $B' = B \wedge \neg(\bot|_B) = B \wedge true = B$. We will now illustrate how iSAT computes a partial interpolant for the partial proof in Fig. 1. Suppose clause c_2 belongs to the clause set A and clause c_3 belongs to B. Further, let $x \in G$, $y \in G$ and $b \in L_B$. In Fig. 2 we decorated the partial proof with the corresponding partial interpolants by applying the construction rules (Definition 2). For the computed partial interpolant $pi(d_4)$ of the clause d_4 we will now show that Lemma 4 is valid. To do so we compute $A' = A \wedge \neg(d_4 \mid_A)$ and $B' = B \wedge \neg(d_4 \mid_B)$. As d_4 does not contain variables from L_A, we know $\neg(d_4 \mid_A) = true$. Because of $\models A \Rightarrow d_2$ ($c_2 \in A$ and $c_2 \rhd d_2$) and $pi(d_4) = d_2$, we conclude $\models A' \Rightarrow pi(d_4)$. In order to show $\models pi(d_4) \Rightarrow \neg B'$ we show that the negation is unsatisfiable $(pi(d_4) \wedge B')$. To obtain B' we have to compute $\neg(d_4 \mid_B) = \neg b \wedge (y \geq 8.5) \wedge (x \leq 1) = f_4$. We know that $\models B' \Rightarrow d_3$ as $c_3 \rhd d_3$ and $c_3 \in B$. It is easy to see that f_4 and d_3 imply $f_5 = (x > 0.31)$. Furthermore, formula part f_4 together with $pi(d_4)$ imply $f_6 = (x \leq 0.31)$ which contradicts f_5 and thus proves the unsatisfiability of $pi(d_4) \wedge B'$.

$$\begin{tikzcd}
F & = & A & \wedge & B & \leftarrow \text{original CNF} \\
 & & \Downarrow & & \Downarrow & \leftarrow \text{deduction rules} \\
 & & A_d \arrow[ddr, dash] & & B_d \arrow[ddl, dash] & \leftarrow \text{derived CNF} \\
 & & & & & \leftarrow \text{resolution rules} \\
 & & & \bot & &
\end{tikzcd}$$

Fig. 3. Motivation

The proof of Lemma 4 is further motivated by the following fact: Suppose iSAT has classified a problem formula as being unsatisfiable by applying resolution. The resolution steps are either performed on clauses containing only simple bounds (e.g. $x > 0.31$) as literals or on simple bounds that have been derived using the deduction rule. To visualize this, Fig. 3 illustrates how iSAT derives the empty clause. Assume $F = A \wedge B$ is unsatisfiable. Then the deduction rule can only be applied to clauses from either A or B. For a clause $cl_{derived}$ derived by the deduction rule, we can state the following:

$$\begin{aligned} &\text{if } c \in A,\ c \rhd cl_{derived} \text{ then } \models A \Rightarrow cl_{derived} \\ &\text{if } c \in B,\ c \rhd cl_{derived} \text{ then } \models B \Rightarrow cl_{derived} \end{aligned} \tag{5}$$

Imagine that through deduction, iSAT was able to derive all clauses needed to produce the empty clause. In Fig. 3, the clause sets A_d and B_d contain all these derived clauses. iSAT derives the empty clause $\bot$ by applying resolution on clauses that are contained in $A_d \cup B_d$. Remember, by definition of the deduction rule, all clauses derived by this rule contain only simple bounds as literals. If we construct a CI I for the two clause sets A_d and B_d, it is also a valid CI for the original defined clause sets A and B. This is because A_d and B_d are over-approximations of A and B. As I is a CI for A_d and B_d we conclude that $\models A_d \Rightarrow I$ and $\models I \Rightarrow \neg B_d$. Together with (5) we conclude $\models A \Rightarrow A_d \Rightarrow I$ and $\models I \Rightarrow \neg B_d \Rightarrow \neg B$. Now we provide a proof sketch of Lemma 4:

Proof Sketch.[2] The proof is similar to that presented in [14]. There, the authors proved that the symmetric construction rules presented by Pavel Pudlàk's algorithm [15] are sound. In our case we have asymmetric rules (similar to McMillan [2]). But it is still possible to prove the invariant of Lemma 4. The proof is given by induction over the depth of a proof tree P computed by iSAT. This makes it necessary to distinguish between different cases and every case itself is proved by showing that the following three properties are valid: (1) For every literal l of $pi(c)$ it holds that $var(l) \in G$, (2) $A' \Rightarrow pi(c)$, and (3) $pi(c) \Rightarrow \neg B'$.

4 iSAT and BMC with Craig Interpolation

As mentioned in Sec. 1, CIs can be applied in BMC. By using CIs, McMillan [2] modified a normal BMC procedure for so called Kripke Structures in such a way

[2] A detailed proof can be found under `http://www.informatik.uni-freiburg.de/~skupfers/tech-report-01.pdf`

that the modified procedure could prove safety properties of a given system. Here, we extend the BMC approach of McMillan to our context so that systems described through rich arithmetic constraints can be verified. Initial experiments with an extension of iSAT as underlying solver illustrate the usefulness of this method. Before presenting the experimental results, we will shortly summarize the main ideas on how BMC can be turned into a proof system.

4.1 Basics

A BMC problem consists of a predicate $\mathit{INIT}(\boldsymbol{x}_0)$ describing the initial state, a predicate $\mathit{TRANS}(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$ defining how variables change from step i to step $i+1$, and lastly a predicate describing unsafe system states $\mathit{TARGET}(\boldsymbol{x}_k)$. A system trace is then defined as follows:

$$\Phi_k = \mathit{INIT}(\boldsymbol{x}_0) \wedge \bigwedge_{i=0}^{k-1} \mathit{TRANS}(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) \wedge \mathit{TARGET}(\boldsymbol{x}_k)$$

Here, the value k is called the *depth*, and the classical BMC approach tries to detect whether or not a system $\mathcal{S}$ can reach an unsafe state at a certain *depth*. This is done by iteratively checking whether $\Phi_0, \Phi_1, \ldots, \Phi_k$ is satisfiable or not. If no failures for large values of k can be found it could be the case that the target state is unreachable for every k. One approach to prove this is done by checking that all reachable states have been proven to be safe. To accomplish this, we can first check the initial reachable states, exactly those described through $\mathit{INIT}(\boldsymbol{x}_0)$. We define the state set that is reachable in exactly k transition steps through:

$$\mathit{REACH}_k = \exists \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{k-1} \mathit{INIT}(\boldsymbol{x}_0) \wedge \bigwedge_{i=0}^{k-1} \mathit{TRANS}(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$$

One way to check whether all states have been explored by normal BMC and searching depth k could be:

$$\mathit{REACH}_k[\boldsymbol{x}_k/\boldsymbol{x}] \Rightarrow \mathit{REACH}_{k-1}[\boldsymbol{x}_{k-1}/\boldsymbol{x}] \vee \cdots \vee \mathit{REACH}_1[\boldsymbol{x}_0/\boldsymbol{x}] \vee \mathit{INIT}(\boldsymbol{x})$$

This is called a fixed-point-check (FPC). The notation $[\boldsymbol{x}_{k-1}/\boldsymbol{x}]$ stands for the substitution of the vector x_{k-1} through the vector x. The check above has the disadvantage of containing several $\exists$-quantifiers which require many quantifier eliminations to be performed in order to solve the formula. There are two main issues associated with quantifier elimination. The first is elimination of quantifiers can lead to an exponential blowup in the size of the formula. The second issue occurs when solving problems that contain transcendental functions (as we do), as there exist no such elimination rules. To obtain an alternative solution, Craig interpolation can be used. We define:

Definition 5 (PREF_l, SUFF_l^k). *Given Φ_k we define PREF_l and SUFF_l^k as:*

$$\mathit{PREF}_l = \mathit{INIT}(\boldsymbol{x}_0) \wedge \bigwedge_{i=0}^{l-1} \mathit{TRANS}(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$$

$$SUFF_l^k = \bigwedge_{i=l}^{k-1} TRANS(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) \wedge \bigvee_{i=k-l}^{k} TARGET(\boldsymbol{x}_k)$$

Here l is the parameter responsible for the number of over-approximated transition steps ($l > 0$). If Φ_k is unsatisfiable, a CI p is computed for the formulae $A = PREF_l$ and $B = SUFF_l^k$. If p implies the initial state, a fixed-point has been reached (i.e. it has been proved that the target state is unreachable). If the initial state is not implied by the CI, we continue increasing the depth k by using p as a new initial state as p is an over-approximation of all states reachable in l steps from the initial state. By setting p as the new initial state the number of added unwindings has been increased by l because p over-approximates all the states reachable in l transition steps. Going on like this could eventually lead to a satisfiable problem formula. This does not mean that the target state is reachable as our initial state is an over-approximation, which can result in a spurious counterexample being detected. In these cases, the solver would then discard the previously calculated CIs, and start a new BMC run at the current unroll depth. For more details please refer to [2]. We want to remark that the challenge in finding a fixed-point depends highly on the generated over-approximations represented by the computed CIs. A problem arising in the context of BMC and Craig interpolation is found in the *strengths* of the computed interpolants. If a CI or over-approximation is to close to the exact reachable states we will have to iterate this procedure many times until a fixed-point is detected. Such CIs are called *strong*. However, if they are too *weak* it can happen that we will often detect counterexamples in the over-approximations. In order to compute interpolants of different strength in the future, we are going to take the slackness information provided by iSAT into account when computing CIs.

4.2 Experiments with iSAT

We implemented the presented approach into the solver iSAT. The data structure used in the iSAT extension stores the partial CIs as a modified *And-Inverter-Graph* (AIG) [5]. This can be done as the CIs in this case are Boolean combinations of simple bounds. Such formulae can be encoded as an equisatisfiable Boolean formula by introducing Boolean variables for each simple bound and further adding constraints that encode the corresponding relations among these simple bounds. The benefit of this data structure is that it supports all Boolean operations needed to construct interpolants. Further, the AIG package can perform satisfiability checks needed in iSAT's FPC routine when applying BMC and Craig interpolation. To get a better picture of how the CIs that iSAT produces look like, we will first give a small two dimensional example. Let A and B be two formulae defined as:

$$\begin{aligned}
A := {} & ((x < 2.5) \Rightarrow (y \geq 2\sin(x))) \\
& \wedge((x \geq 2.5 \wedge x < 5) \Rightarrow (y \geq 0.125x^2 + 0.41)) \\
& \wedge((x \geq 5 \wedge x \leq 6) \Rightarrow (y \geq -0.5x + 6.04)) \\
B := {} & ((x < 3) \Rightarrow (y \leq -0.083 + (x\cos(0.1\exp(x))))) \\
& \wedge((x \geq 3 \wedge x \leq 6) \Rightarrow (y \leq -x^2 + 10x - 22.35))
\end{aligned} \tag{6}$$

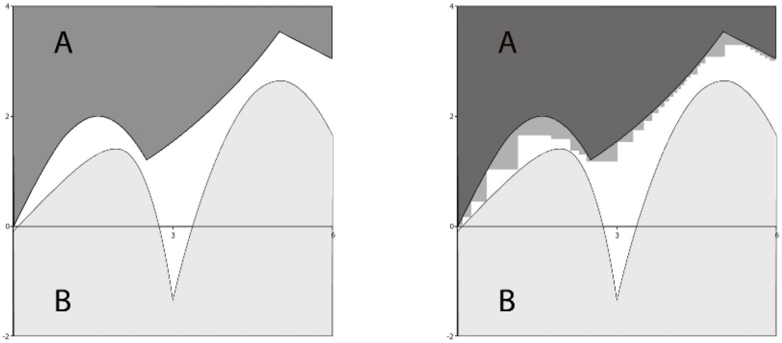


Fig. 4. On the left side you see two formulae A and B with the property that $A \wedge B$ is unsatisfiable. On the right side a CI for the formula pair (A, B) is pictured. The CI has been computed by using iSAT.

Using the initial bounds $x \in [0, 6]$ and $y \in [-2, 4]$, the problem can be visualized on the left hand side of Fig. 4. In this figure, the region where the formula A (B) is satisfied is labeled with **A** (**B**). It is quite obvious that $A \wedge B$ is unsatisfiable as the intersection of the two regions is empty, and iSAT is easily able to find an AB-refutation. A CI which is generated on-the-fly can be seen on the right hand side of Fig. 4. The CI ci covers the region of A and is thus implied by A. As ci has an empty intersection with B it directly follows that $ci \wedge B$ is unsatisfiable. The shape of the interpolant is a combination of boxes. This is explained by the construction rules and the fact that iSAT only performs resolution on clauses containing simple bounds as literals.

To show the usefulness of the CIs that iSAT can produce, we studied six different BMC benchmarks together with some valid safety properties. The transition relations of these benchmarks contain non-linear and linear equations. Of course, even our approach is not designed for pure linear systems, it should in principle work for such systems. To show this, we modeled two linear systems presented by Alur et al. in [16]. The first system describes a *thermostat* and the second one is a version of a *leaking gas burner*.

The first non-linear problem is called *the logistic map* [17] and is a polynomial mapping of degree 2. Mathematically, the logistic map is written as $x_{n+1} = r \cdot x_n(1 - x_n)$ where x_n is a number between zero and one. This map illustrates chaotic behaviour, but can exhibit periodic behavior by setting $r = 3.2$. When $r = 3.2$, the logistic map oscillates between two values, and we defined the safety regions to be $(0.78 \leq x \wedge x \leq 0.82) \vee (0.48 \leq x \wedge x \leq 0.52)$ (approx. 0.8 and 0.5).

The next example is the Hènon map [18], a chaotic map introduced by Michel Hènon and mathematically defined as $x_{n+1} = y_n + 1 - ax_n^2$ and $y_{n+1} = bx_n$. The map depends on two parameters a and b. Setting $a = 1.25$ and $b = 0.3$ makes the Hènon map oscillating between seven different values. The safety properties for these maps are the disjunction of small intervals containing the periodic values in a similar fashion to the logistic map case.

Table 1. Results

Benchmark	**#Decisions**	**#Deductions**	**#AigNodes**	**Depth**	**Time FPC**	**Time**
hènon	6797	13799353	10260	283	0.8	45.26
logistic	2085	596984	3187	60	0.05	2.02
accelerate	13	1809	88	7	0.00	0.02
cruise control	384	99841	1960	55	0.01	0.27
thermostat	346	93855	105291	6	0.19	1.18
gas burner	6189	3439885	34105	21	0.31	24.12

Next, we consider two BMC problems describing an accelerating car. Taking the air resistance into account, the relationship between the car's velocity and the physical drag contains quadratic functions. The first benchmark describes the velocity of a car that is accelerating with constant force. Due to the air resistance the car cannot drive faster than $49.61\frac{m}{sec}$ which is our safety property. The initial state (INIT) and the transition relation (TRANS) of this hybrid system are:

$$INIT := \begin{array}{ll} v_{car_0} = 0 \wedge & // \text{ velocity at step 0} \\ F_{res_0} = 1000 \wedge & // \text{ resultant force at step 0} \\ a_{car_0} = 0.0005 \cdot F_{res_0} & // \text{ acceleration at step 0} \end{array} \tag{7}$$

The transition relation computes the resultant force $F_{res_{i+1}}$ at time step $i+1$. In order to compute $F_{res_{i+1}}$ we need to compute F_{air_i} first. In this example the interaction caused by the drag simplifies to $F_{air_i} = 0.40635 \cdot v_{car_i}^2$.

$$TRANS := \begin{array}{ll} F_{res_{i+1}} = 1000 - F_{air_i} \wedge & // \text{ resultant force at step } i+1 \\ F_{air_i} = 0.5418 \cdot v_{car_i}^2 \wedge & // \text{ drag force at step } i \\ a_{car_{i+1}} = 0.0005 \cdot F_{res_{i+1}} \wedge & // \text{ acceleration at step } i+1 \\ v_{car_{i+1}} = v_{car_i} + a_{car_i} & // \text{ velocity at step } i+1 \end{array} \tag{8}$$

We also extended the above example by adding a controller that is responsible for accelerating the car. By doing this we end up with a simplified cruise control system. The job of the controller is to maintain a certain velocity by either accelerating with a constant force, or applying no force at all. The safety property in this example is that the velocity is between $9.9\frac{m}{sec}$ and $10.1\frac{m}{sec}$ (additionally, we set the velocity in the initial state to 10, i.e. $v_{car_0} = 10$).

The results of our work with iSAT and Craig interpolation can be seen in Table 1. The test machine used for the results stated here has a Quadcore Intel Q9450 processor @ 2.66GHz.

The columns #**Decisions** and #**Deductions** show the number of decisions and deductions iSAT needs to solve the entire problem. The number of internal AIG-nodes needed to store the partial CIs is given in column #**Aig-Nodes**. The unroll depth where the FPC was successful is provided in column **Depth**. The last two columns provide information about the time needed for performing the FPC (**Time FPC**) and the overall time (**Time**). The examples presented demonstrate that it is possible to successfully apply Craig interpolation in the case of systems containing non-linear behaviour, and the time needed to solve the FPC is negligible. Further the column #**Aig-Nodes** shows that the approximate size of the CIs stays relatively small in all cases. Concerning the Hènon Map where the property and the behaviour are very complex, the overall time is increasing as many CIs have to be computed until the FPC is successful, in this case the solving depth is 283. To further demonstrate the efficacy of Craig interpolation for iSAT we also considered the behaviour of "pure" iSAT on the benchmarks given above.In the case of the Hènon map iSAT obtains unsat for BMC unrolling depth $k < 158$ and terminates with a candidate solution at unroll depth 158. In contrast to this, using CIs in iSAT helps to prove the unsatisfiability for all k and thus prove safety. In order to find a fixed-point we had to modify certain parameters that influence the overall search procedure. It seems to be profitable to modify the iSAT variable decision heuristic to first decide Boolean simple bounds, next real simple bounds, and in the end iSAT is allowed to split certain intervals. Besides different decision heuristics, one can generate different CIs by changing the minimal splitting width. At the moment we do not take slackness information into account when computing CIs. Suppose you want to verify that a variable $x \geq 10$ cannot reach any negative value when divided again and again by 2. A possible CI representing those states that are reachable in one system step could look like $x \geq 5$. But you could also compute a CI with $x \geq 0$. If the later CI becomes the new initial state for the next iteration one could achieve the same CI again, and thus implies that we found a fixed-point. In order to compute such CIs we will have to take slackness information during resolution steps into account and influence the computation of CIs accordingly.

5 Conclusion

In this paper we introduced a method to generate CIs for formulae containing non-linear equations. Furthermore, we implemented our approach in the SMT solver iSAT, which is based on interval arithmetic embedded in a DPLL framework allowing it to reason about linear and non-linear constraints. To the best of our knowledge this is the first approach to compute CIs for arbitrary formulae containing Boolean combinations of linear and non-linear constraints. We showed that the CIs constructed can be used to verify safety properties, extending the work done by McMillan. Currently, we study heuristics to strengthen CIs for a given formula pair (A, B), e.g. by exploiting slackness between different constraints. We are also integrating a linear program solver (LP-solver) to combine LP-solving and iSAT's Craig interpolation and thus increase the performance of our solver for systems containing a large number of linear constraints.

References

1. Craig, W.: Linear reasoning: A new form of the Herbrand-Gentzen theorem. Journal of Symbolic Logic (3), 250–268 (1957)
2. McMillan, K.L.: Interpolation and SAT-based Model Checking. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003)

3. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. (1) (2005)
4. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software Model Checker Blast: Applications to Software Engineering. International Journal on Software Tools for Technology Transfer (STTT) (5-6), 505–525 (2007)
5. Scholl, C., Disch, S., Pigorsch, F., Kupferschmid, S.: Using an SMT solver and Craig interpolation to detect and remove redundant linear constraints in representations of non-convex polyhedra. In: International Workshop on Satisfiability Modulo Theories, pp. 18–26 (2008)
6. Cimatti, A., Griggio, A., Sebastiani, R.: Efficient Generation of Craig Interpolants in Satisfiability modulo theories. In: CoRR (2009)
7. Fränzle, M., Herde, C., Ratschan, S., Schubert, T., Teige, T.: Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure. JSAT Special Issue on Constraint Programming and SAT, 209–236 (2007)
8. Matiyasevich, Y.V.: Enumerable sets are Diophantine. Soviet Mathematics. Doklady 11(2), 354–358 (1970)
9. Davis, M., Logemann, G., Loveland, D.: A Machine Program for Theorem Proving. In: CACM, pp. 394–397 (1962)
10. Davis, M., Putnam, H.: A Computing Procedure for Quantification Theory. Journal of the ACM (3), 201–215 (1960)
11. Benhamou, F., Granvilliers, L.: Continuous and Interval Constraints. In: Handbook of Constraint Programming. Foundations of Artificial Intelligence, pp. 571–603 (2006)
12. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic, Part 2, 115–125 (1970)
13. Kupferschmid, S., Becker, B., Teige, T., Fränzle, M.: Proof certificates and non-linear arithmetic constraints. In: IEEE Design and Diagnostics of Electronic Circuits and Systems. IEEE, Los Alamitos (2011)
14. Yorsh, G., Musuvathi, M.: A Combination Method for Generating Interpolants. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 353–368. Springer, Heidelberg (2005)
15. Pudlàk, P.: Lower bounds for resolution and cutting planes proofs and monotone computations. J. of Symbolic Logic, 981–998 (1995)
16. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138, 3–34 (1995)
17. May, R.M.: Simple Mathematical Models with very Complicated Dynamics. Nature, 459 (1976)
18. Hènon, M.: A two-dimensional mapping with a strange attractor. In: Communications in Mathematical Physics (1976)

On the Verification of Timed Ad Hoc Networks

Parosh Aziz Abdulla[1], Giorgio Delzanno[2], Othmane Rezine[1],
Arnaud Sangnier[3,⋆], and Riccardo Traverso[2]

[1] Uppsala University
[2] University of Genova
[3] Liafa-University Paris 7

Abstract. We study decidability and undecidability results for parameterized verification of a formal model of timed Ad Hoc network protocols. The communication topology is represented by a graph and the behavior of each node is represented by a timed automaton communicating with its neighbors via broadcast messages. We consider verification problems formulated in terms of reachability, starting from initial configurations of arbitrary size, of a configuration that contain at least one occurrence of a node in a certain state. We study the problem for dense and discrete time and compare the results with those obtained for (fully connected) networks of timed automata.

1 Introduction

In recent years there has been an increasing interest in analysis and verification methods for ad hoc networks, see e.g. [7,8,11,14,15,16]. Ad hoc networks consist of collections of wireless hosts that communicate with their neighbors by sending broadcast messages. In this context, protocols are supposed to work independently from a specific configuration of the network. Indeed, discovery protocols are often applied in order to identify the vicinity of a given node. In the AHN model proposed in [7] undirected graphs are used to represent a network in which each node executes an instance of a fixed (untimed) interaction protocol. Since individual nodes are not aware of the network topology, it is natural to consider here verification problems in which the size and shape of the initial configuration is not fixed a priori. For an untimed model of ad hoc network protocols, decidability and undecidability of this kind of parameterized verification problems are studied in [7]. We observe however that protocols for ad hoc networks are often based on time-sensitive conditions like time-outs and timestamps added to flooded data. Parameterized verification of timed automata has been studied for fully connected networks with rendez-vous communication (Timed Networks), e.g., in [3,4,5,6].

A natural combination of the AHN model in [7] and of Timed Networks is obtained by adding a connectivity graph to a network of timed automata communicating via broadcast messages, we call the resulting model Timed Ad hoc Networks (TAHNs). For a fixed initial configuration, TAHNs can be specified in

⋆ This author is partially supported by ANR ImpRo (ANR-10- BLAN-0317).

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 256–270, 2011.

model checkers like Uppaal [17] by using a shared global matrix to specify the communication topology. This idea has been used to verify safety properties of the LMAC protocol for initial configurations with a small number of nodes [11].

Following [7,8], in this paper we study decidability and undecidability properties of the local state reachability problem parametric on the initial configuration of a TAHN, i.e., the problem of checking the existence of an initial configuration that can evolve using continuous and discrete steps into a configuration exposing a given local state – usually representing an error.

Compared to the positive results obtained for Timed Networks where the control state reachability problem is decidable for processes with a single clock, for the same type of processes the problem becomes undecidable in a very simple class of topologies in which nodes are connected so as to form stars with diameter five. The undecidability result can be ported to the more general class of graphs with bounded path (for some bound $N \geq 5$ on the length – number of nodes – of paths). In the untimed case local state reachability is decidable for bounded path topologies [7]. Furthermore, the problem turns out to be undecidable in the class of cliques of arbitrary order (that contains graphs with arbitrarily long paths) in which each timed automaton has at least two clocks. Decidability holds for special topologies like stars with diameter three and cliques of arbitrary order assuming that the timed automaton in each node has a single clock (as in Timed Networks).

For discrete time, we show that the local state reachability problem becomes decidable for processes with any number of clocks in the class of graphs with bounded path. The same holds for cliques of arbitrary order as in the case of dense time.

Related Work. Decidability issues for untimed models of ad hoc Networks have been considered in [7,8]. Abstraction techniques for untimed selective broadcast protocols has been considered in [14]. Model checking for timed automata has been applied to verify protocols for ad hoc networks with a fixed number of nodes in [11]. Models with a discrete global clock and a lazy exploration of configurations of fixed size has been considered in [16]. Formal specification languages for timed models of ad hoc networks have been proposed, e.g., in [13]. In contrast to these works, we consider computatibility issues for verification of timed ad hoc networks with parametric initial configurations. Decidability of some cases is proved by resorting to an extension of Timed Networks with transfers. In the untimed case the combination of rendez-vous and transfers is considered in Datanets, a model in which processes have data taken from an ordered domain [12].

Outline. In the next section we give some preliminaries. The models of Timed Ad Hoc Networks and Time Networks are introduced in Sections 3–4. In Section 5, we introduce the undecidability results for three different topologies. We give decidability for different topologies under the dense time semantic in Section 6 and under the discrete time semantics in Section 7. Finally, we give some conclusions in Section 8.

Detailed proofs are given in [2].

2 Preliminaries

We use $\mathbb{N}$ and $\mathbb{R}^{\geq 0}$ for the set of natural numbers and set of non-negative real numbers respectively. For sets A and B, we use $f : A \mapsto B$ to denote that f is a total function that maps A to B. For $a \in A$ and $b \in B$, we use $f[a \hookleftarrow b]$ to denote the function f' defined as follows: $f'(a) = b$ and $f'(a') = f(a')$ for all $a' \neq a$. We use $[A \mapsto B]$ to denote the set of all total functions from A to B.

3 Timed Ad Hoc Networks

An ad hoc network consists of a graph where the nodes represent processes that run a common predefined protocol. This protocol is defined by a communicating timed automaton. The values of the clocks of the automata inside the processes are incremented continuously all at the same rate. In addition, a process may perform a discrete transition. The latter is either a local transition or the result of a communication event. In a *local* transition, the process changes its local state without interacting with the other processes. Communication is performed through *selective broadcast*, where a process sends a broadcast message to the network. The effect of a broadcast is local to the vicinity of the sender, i.e., only the neighbors of the sending process are able to receive the message. The connectivity of the nodes is reflected by the edges of the graph. Furthermore, transitions are conditioned by values of the clocks of the process, and may reset the values of some clocks.

We assume that each process operates on a set X of clocks. A *guard* is a Boolean combination of predicates of the form $k \lhd x$ for $k \in \mathbb{N}$, $\lhd \in \{=, <, \leq, >, \geq\}$, and $x \in X$. A *reset* R is a subset of X. We will use guards to impose conditions on the clocks of processes that participate in transitions, and use resets to identify the clocks that will be reset during the transition. A *clock valuation* is a mapping $\mathtt{X} : X \mapsto \mathbb{R}^{\geq 0}$. For a guard g and a clock valuation $\mathtt{X}$, we write $\mathtt{X} \models g$ to indicate the validity of the formula we get by replacing each clock x in g by its value $\mathtt{X}(x)$. Also, we will assume a finite alphabet Σ. The alphabet induces a set of events, where an event is of one of three forms: (i) *empty event* τ that represents a local move; (ii) *broadcast event* $!!a$, with $a \in \Sigma$, that represents broadcasting the message a; or (ii) *receive event* $??a$ with $a \in \Sigma$, that represents receiving the message a (that has been broadcast by another process).

Formally, a *Timed Ad Hoc Network* (*TAHN* for short) is defined by a pair $\mathcal{T} = (G, P)$. The first component $G = (V, E)$ is graph where V is a finite set of vertices and $E \subseteq V \times V$ is a set of edges. The second component P, called the *protocol*, is a pair $(Q, \mathcal{R})$ where Q is a finite set of *states*, and $\mathcal{R}$ is a finite set of *rules*. Intuitively, the graph G defines the topology of $\mathcal{T}$ where the set V represents the nodes, and E defines the connectivity of the nodes. The vertices belonging to an edge are called the *endpoints* of the edge. For an edge $(u, v) \in E$, we often use the notation $u \sim v$ and say that the vertices u and v are adjacent to each other. Furthermore, P defines the protocol that runs inside the nodes, where Q is the set of *local states* of each node, while $\mathcal{R}$ is a set of *rules* describing the behavior of each node. A rule $\rho \in \mathcal{R}$ is of the form $\left(q, g \xrightarrow{e} R, q'\right)$ where $q, q' \in Q$, g is a guard, e is an event, and R is a reset. We use $\mathtt{CLOCKS}(P)$ to denote that number of clocks inside each process.

Configurations. A configuration γ is a pair $(\mathcal{Q}, \mathcal{X})$, where $\mathcal{Q} : V \mapsto Q$ and $\mathcal{X} : V \mapsto [X \to \mathbb{R}^{\geq 0}]$, i.e., the configuration assigns to each node a local state and assigns to each clock in the node a value (in $\mathbb{R}^{\geq 0}$).

Transition Relation. For configurations $\gamma = (\mathcal{Q}, \mathcal{X})$ and $\gamma' = (\mathcal{Q}', \mathcal{X}')$, we write $\gamma \Longrightarrow_{\mathcal{T}} \gamma'$ to denote that one of the following conditions is satisfied:

- *Local:* There exists a rule $\rho = \left(q, g \xrightarrow{\tau} R, q'\right)$ and a vertex $v \in V$ such that $\mathcal{Q}(v) = q$, $\mathcal{X}(v) \models g$, $\mathcal{Q}' = \mathcal{Q}\left[v \hookleftarrow q'\right]$, and $\mathcal{X}' = \mathcal{X}\left[v \hookleftarrow \mathtt{X}\right]$ where $\mathtt{X}(x) = 0$ if $x \in R$ and $\mathtt{X}(x) = \mathcal{X}(v)(x)$ otherwise.
- *Broadcast:* There exists a rule $\left(q, g \xrightarrow{!!a} R, q'\right)$ and a vertex $v \in V$ such that $\mathcal{Q}(v) = q$, $\mathcal{X}(v) \models g$, $\mathcal{Q}'(v) = q'$, $\mathcal{X}'(v)(x) = 0$ if $x \in R$, and $\mathcal{X}'(v)(x) = \mathcal{X}(v)(x)$ otherwise. Furthermore, for each $v_1 \in V - \{v\}$, one of the following conditions is satisfied:
  - $v_1 \sim v$ and there is a rule of the form $\left(q_1, g_1 \xrightarrow{??a} R_1, q_1'\right)$ such that $\mathcal{Q}(v_1) = q_1$, $\mathcal{X}(v_1) \models g_1$, $\mathcal{Q}'(v_1) = q_1'$, $\mathcal{X}'(v_1)(x) = 0$ if $x \in R_1$, and $\mathcal{X}'(v_1)(x) = \mathcal{X}(v_1)(x)$ otherwise.
  - $\mathcal{Q}'(v_1) = \mathcal{Q}(v_1)$, $\mathcal{X}'(v_1) = \mathcal{X}(v_1)$, and either $v_1 \not\sim v$ or there is no rule of the form $\left(q_1, g_1 \xrightarrow{??a} R_1, q_1'\right)$ for any g_1, R_1, q_1' with $\mathcal{Q}(v_1) = q_1$ and $\mathcal{X}(v_1) \models g_1$.
- *Time:* There is a $\delta \in \mathbb{R}^{\geq 0}$ such that $\mathcal{Q}(v') = \mathcal{Q}(v)$ and $\mathcal{X}'(v)(x) = \mathcal{X}(v)(x) + \delta$ for all $v \in V$ and $x \in X$.

Topology. A *topology Top* restricts the shape of the underlying graph $G = (V, E)$ in a TAHN $\mathcal{T} = (P, G)$. We write $G \in \mathit{Top}$ to indicate that G satisfies *Top*. Below, we give examples of some topologies.

- We denote by $\mathtt{GRAPH}$ the topology consisting of all finite graphs.
- The star topology of depth ℓ (with $\ell \geq 0$), denoted $\mathtt{STAR}(\ell)$, characterizes graphs G for which there is a partitioning $\{v_0\} \cup V_1 \cup \cdots \cup V_\ell$ of V such that (i) $v_0 \sim v_1$ for all $v_1 \in V_1$, (ii) for each $1 \leq i < \ell$ and $v_i \in V_i$ there is exactly one $v_{i+1} \in V_{i+1}$ with $v_i \sim v_{i+1}$, (iii) for each $1 < i \leq \ell$ and $v_i \in V_i$ there is exactly one $v_{i-1} \in V_{i-1}$ with $v_i \sim v_{i-1}$, and (iv) no other nodes are adjacent to each other. In other words, in a star topology of dimension ℓ, there is a central node v_0 and an arbitrary number of *rays*. A ray consists of a sequence of ℓ nodes, starting from v_0 followed by some $v_1 \in V_1$, and then by some $v_2 \in V_2$, etc. We call v_0 the *root*, call the nodes in $V_1, \ldots, V_{\ell-1}$ the *internal nodes*, and call the nodes in V_ℓ the *leaves* of G.
- The bounded path topology of bound ℓ (with $\ell \geq 0$), denoted $\mathtt{BOUNDED}(\ell)$, characterizes graphs G for which the length of the maximal simple path in

G is bounded by ℓ. This means that there does not exist a finite sequence of vertices $(v_i)_{1 \leq i \leq m}$ satisfying the following conditions (1) $m > \ell$, (2) $v_i \neq v_j$ for all i, j in $\{1, \ldots, m\}$ such that $i \neq j$ and (3) $v_i \sim v_{i+1}$ for all i such that $1 \leq i < m - 1$.

- The set of cliques, denoted `CLIQUE` characterizes graphs G where $v_1 \sim v_2$ for all $v_1, v_2 \in V$ with $v_1 \neq v_2$.

Reachability. We assume a distinguished local state $q^{init} \in Q$. A configuration γ^{init} is said to be *initial* if it is of the form $\left(\mathcal{Q}_Q^{init}, \mathcal{X}^{init}\right)$ where $\mathcal{Q}_Q^{init}(v) = q^{init}$, and $\mathcal{Q}_Q^{init}(v)(x) = 0$ for all $v \in V$ and $x \in X$. In other words, all the processes are in their initial local states and all the clocks have value 0. A *computation* π of $\mathcal{T}$ is a sequence $\gamma_0 \Longrightarrow_{\mathcal{T}} \gamma_1 \Longrightarrow_{\mathcal{T}} \cdots \Longrightarrow_{\mathcal{T}} \gamma_n$ where γ_0 is an initial configuration. In such a case, we say that γ_n is *reachable* in $\mathcal{T}$. For a local state $q \in Q$, we say that q is *reachable* in $\mathcal{T}$ if there is a configuration $\gamma = (\mathcal{Q}, \mathcal{X})$ such that γ is reachable in $\mathcal{T}$ and $\mathcal{Q}(v) = q$ for some $v \in V$.

The *(local state) reachability problem* for a topology *Top* and a number K, denoted $\texttt{TAHN}-\texttt{Reach}(\mathit{Top}, K)$, is defined as follows:

> Given a protocol P with $\texttt{CLOCKS}(P) = K$ and a local state $q \in Q$, is there a TAHN $\mathcal{T} = (P, G)$ such that $G \in \mathit{Top}$ and q is reachable in $\mathcal{T}$.

The following results concerning untimed Ad Hoc Networks have been proved.

Theorem 1 ([7]). $\texttt{TAHN}-\texttt{Reach}(\texttt{GRAPH}, 0)$ *is undecidable. For each* $K \geq 0$*, the problem* $\texttt{TAHN}-\texttt{Reach}(\texttt{BOUNDED}(K), 0)$ *is decidable.*

4 Timed Networks

In this section, we recall the model of *Timed Networks* (*TN* for short) [6]. In a similar manner to TAHNs, a TN contains an arbitrary number of identical *timed processes* that operate on a finite number of local real-valued clocks. However, there are three main differences between TNs and TAHNs. First, a TN contains a distinguished *controller* that is a finite-state automaton without any clocks[1]. Second, each process in a TN may communicate with all other processes and hence it is not meaningful to describe topologies in the case of TNs. Finally, communication takes place through rendez-vous between fixed sets of processes rather than broadcast messages. In a similar manner to TAHNs, the values of all clocks in a TN are incremented continuously at the same rate. In addition, the TN can change its configuration according to a finite number of *rules*. Each rule describes a set of transitions in which the controller and a fixed number of processes synchronize and simultaneously change their states. A rule may be conditioned on the local state of the controller, together with the local states and clock values of the processes. If the conditions for a rule are satisfied, then a

[1] This is the model defined in [6]. Adding clocks to the controller does not affect the decidability results.

transition may be performed where the controller and each participating process changes its state. During a transition, a process may reset some of its clocks to 0.

We assume a finite set X of clocks and define guards and resets in a similar manner to TAHNs. A *family of timed networks* (*timed network* for short) $\mathcal{N}$ is a pair $(Q, \mathcal{R})$, where Q is a finite set of *states*, partitioned into a set Q^{ctrl} of *controller states*, and a set Q^{proc} of *process states*; and $\mathcal{R}$ is a finite set of *rules* where each rule is of the form

$$[q_0 \rightarrow q_0'] \; [q_1; g_1 \rightarrow R_1; q_1'] \cdots \; [q_n; g_n \rightarrow R_n; q_n']$$

such that $q_0, q_0' \in Q^{ctrl}$, for all $i : 1 \leq i \leq n$ we have: $q_i, q_i' \in Q^{proc}$, g_i is a guard, and R_i is a reset. Intuitively, the set Q^{ctrl} represents the states of the controller and the set Q^{proc} represents the states of the processes. A rule of the above form describes a set of transitions of the network. The rule is enabled if the state of the controller is q_0 and if there are n processes with states $q_1, \cdots, q_n$ whose clock values satisfy the corresponding guards. The rule is executed by simultaneously changing the state of the controller to q_0' and the states of the n processes to $q_1', \cdots, q_n'$, and resetting the clocks belonging to the sets $R_1, \ldots, R_n$.

Configurations. A configuration γ of a timed network $(Q, \mathcal{R})$ is a tuple of the form $(I, q, \mathcal{Q}, \mathcal{X})$, where I is a finite *index set*, $q \in Q^{ctrl}$, $\mathcal{Q} : I \rightarrow Q^{proc}$, and $\mathcal{X} : I \rightarrow X \rightarrow \mathbb{R}^{\geq 0}$. Intuitively, the configuration γ refers to the controller whose state is q, and to $|I|$ processes, whose states are defined by $\mathcal{Q}$. The clock values of the processes are defined by $\mathcal{X}$. More precisely, for $i \in I$ and $x \in X$, $\mathcal{X}(i)(x)$ gives the value of clock x in the process with index i. We use $|\gamma|$ to denote the number of processes in γ, i.e., $|\gamma| = |I|$.

Transition Relation. The timed network $\mathcal{N}$ above induces a transition relation $\longrightarrow_{\mathcal{N}}$ on the set of configurations. The relation $\longrightarrow_{\mathcal{N}}$ is the union of a *discrete* transition relation $\longrightarrow_D$, representing transitions induced by the rules, and a *timed* transition relation $\longrightarrow_T$ which represents passage of time.

The discrete relation $\longrightarrow_D$ is the union $\bigcup_{r \in \mathcal{R}} \longrightarrow_r$, where $\longrightarrow_r$ represents a transition performed according to rule r. Let r be a rule of the form described in the above definition of timed networks. Consider two configurations $\gamma = (I, q, \mathcal{Q}, \mathcal{X})$ and $\gamma' = (I, q', \mathcal{Q}', \mathcal{X}')$. We use $\gamma \longrightarrow_r \gamma'$ to denote that there is an injection $h : \{1, \ldots, n\} \rightarrow I$ such that for each $i : 1 \leq i \leq n$ we have:

1. $q = q_0$, $\mathcal{Q}(h(i)) = q_i$, and $\mathcal{X}(h(i)) \models g_i$. That is, the rule r is enabled.
2. $q' = q_0'$, and $\mathcal{Q}'(h(i)) = q_i'$. The states are changed according to r.
3. If $x \in R_i$ then $\mathcal{X}'(h(i))(x) = 0$, while if $x \notin R_i$ then $\mathcal{X}'(h(i))(x) = \mathcal{X}(h(i))(x)$. In other words, a clock is reset to 0 if it occurs in the corresponding set R_i. Otherwise its value remains unchanged.
4. $\mathcal{Q}'(j) = \mathcal{Q}(j)$ and $\mathcal{X}_k'(j) = \mathcal{X}_k(j)$, for $j \in I \setminus \text{range}(h)$, i.e., the process states and the clock values of the non-participating processes remain unchanged.

A *timed transition* is of the form $\gamma \longrightarrow_{T=\delta} \gamma'$ where $\gamma = (I, q, \mathcal{Q}, \mathcal{X})$, $\delta \in \mathbb{R}^{\geq 0}$, $\gamma' = (I, q, \mathcal{Q}, \mathcal{X}')$, $\mathcal{X}'(j)(x) = \mathcal{X}(j)(x) + \delta$ for all $j \in I$ and $x \in X$. We use $\gamma \longrightarrow_T \gamma'$ to denote that $\gamma \longrightarrow_{T=\delta} \gamma'$ for some $\delta \in \mathbb{R}^{\geq 0}$.

We define $\longrightarrow_{\mathcal{N}}$ to be $\longrightarrow_D \cup \longrightarrow_T$ and use $\overset{*}{\longrightarrow}_{\mathcal{N}}$ to denote the reflexive transitive closure of $\longrightarrow_{\mathcal{N}}$. Notice that if $\gamma \longrightarrow_{\mathcal{N}} \gamma'$ then the index sets of γ and γ' are identical and therefore $|\gamma| = |\gamma'|$. For a configuration γ and a controller state q, we use $\gamma \overset{*}{\longrightarrow}_{\mathcal{N}} q$ to denote that there is a configuration γ' of the form $(I', q, \mathcal{Q}', \mathcal{X}')$ such that $\gamma \overset{*}{\longrightarrow}_{\mathcal{N}} \gamma'$.

Given $\gamma_0 \longrightarrow_{\mathcal{N}} \gamma_1 \longrightarrow_{\mathcal{N}} \gamma_2 \ldots \longrightarrow_{\mathcal{N}} \gamma_n$, we say that $\gamma_0, \ldots, \gamma_n$ is a computation of $\mathcal{N}$.

Reachability. We assume a distinguished initial controller state $q^{init}_{ctrl} \in Q^{ctrl}$ and a distinguished initial process state $q^{init}_{proc} \in Q^{proc}$. A configuration $\gamma^{init} = (I, q, \mathcal{Q}, \mathcal{X})$ is said to be *initial* if $q = q^{init}_{ctrl}$, $\mathcal{Q}(i) = q^{init}_{proc}$, and $\mathcal{X}(i)(x) = 0$ for each $i \in I$ and $x \in X$. This means that an execution of a timed network starts from a configuration where the controller and all the processes are in their initial states, and the clock values are all equal to 0. Notice that there is an infinite number of initial configurations, namely one for each index set I. Concepts such as that of computations and reachability are extended from TAHNs to TNs in the obvious way.

The *(controller state) reachability problem* $\texttt{TN}-\texttt{Reach}(K)$ is defined by a timed network $(Q, \mathcal{R})$ with K clocks (in each process), and a controller state q. The task is to check whether q is reachable or not. In [3], the following result is shown.

Theorem 2. $\texttt{TN}-\texttt{Reach}(2)$ *is undecidable.*

5 Undecidability with Dense Time

In this section, we show undecidability of the reachability problem for three classes of TAHNs, namely (i) those with *star* topologies of depth 2 (one root and several rays with two nodes) and with a single clock in each node; (ii) those with *clique* topologies provided that each node has two clocks; and (iii) those with *bounded path* topologies if the length of simple paths of the underlying graph is at least 5 and with a single clock in each node. In the first two cases, the undecidability result is shown through a reduction from $\texttt{TN}-\texttt{Reach}(2)$ (that is undecidable by Theorem 2). The main idea of the proofs is to show that we can simulate rendez-vous (the communication model of TNs) by broadcast (the communication model of TAHNs) In each case we will simulate a TN $\mathcal{N}$ with two clocks per process by a TAHN $\mathcal{T}$. We will refer to the clocks inside a process of $\mathcal{N}$ as x_1 and x_2 respectively. For each state q in $\mathcal{N}$, we will have a corresponding state $\mathcal{T}(q)$ in $\mathcal{T}$. Furthermore, we will have a number of auxiliary states in $\mathcal{T}$ that we need to perform the simulation. The third undecidability result (for bounded path topologies) is shown by simulating the star case.

Two-Star Topologies. We show that the reachability problem for the star topology is undecidable even when the rays are restricted to have depth 2 and the nodes are restricted to have a single clock.

Theorem 3. $\texttt{TAHN}-\texttt{Reach}(\texttt{STAR}(2), 1)$ *is undecidable.*

Given a TN $\mathcal{N} = (Q, \mathcal{R})$ and a controller state q in $\mathcal{N}$, we define a TAHN $\mathcal{T} = (P, G)$ such that $G \in \texttt{STAR}(2)$ and $\texttt{CLOCKS}(P) = 1$, together with a local state $\mathcal{T}(q)$, such that q is reachable in $\mathcal{T}$ iff q is reachable in $\mathcal{N}$. The root of $\mathcal{T}$ plays the role of the controller in $\mathcal{N}$. Furthermore, each ray in $\mathcal{T}$ plays the role of one process in $\mathcal{N}$. The local state of a process in $\mathcal{T}$ is stored in the internal node of the corresponding ray. Furthermore, the two clocks x_1, x_2 of a process are represented by the clock of the internal node resp. the clock of the leaf of the ray. For technical reasons, we require that $\mathcal{T}$ has at least three rays. In case $\mathcal{N}$ has fewer than three processes, the additional rays will not simulate any processes, and remain passive (except during the initialization phase; see below). The simulation consists of two phases.

Initialization. Recall that the nodes of a TAHN are identical in the sense that they execute the same (predefined) protocol. This means that the nodes are not *a priori* aware of their positions inside the network. The purpose of the initialization phase is to identify the nodes that play the roles of the controller and those that play the roles of the different processes. First, a node may broadcast a message where it requests to become the node that simulates the controller in $\mathcal{N}$. In order to succeed, P requires that it should receive acknowledgements from at least three other processes.

Notice that only the root of $\mathcal{T}$ can be successful since it is the only node that is connected to more than two other nodes (the internal nodes are connected to two other nodes while the leafs are connected to only one other node).

Once the root has become the controller, it will make the internal nodes aware of their positions. It does that by sending a broadcast message. Due to the star topology, this message is received only by the internal nodes. A node receiving this broadcast message will initiate a "local protocol" inside its ray as follows: (i) It changes local state to reflect that it now knows that it is indeed an internal node. (ii) It makes the leaf of the ray aware of its position by broadcasting a message. Such a message is received only by the leaf of the ray and by the root (the latter simply ignores the message). (iii) The leaf broadcasts an acknowledgment (that can only be received by the internal node of the ray). (iv) The internal node changes state when it receives the acknowledgement and declares itself ready for the next step. Notice that the internal node and the leaf may choose to ignore performing steps (ii) or (iv). In such a case we say that the ray has "failed", otherwise we declare the ray to be "successful". In the last step of the initialization, the root will send one more broadcast where the following takes places: (i) It changes local state to $\mathcal{T}(q_{ctrl}^{init})$ which means that it is now simulating the initial controller state. (ii) It checks that its clock is equal to 0 which means that the initialization phase has been performed instantaneously. (iii) The internal nodes of the successful rays will change state to $\mathcal{T}(q_{proc}^{init})$. The rest of the nodes will remain passive throughout the rest of the simulation. Now all the nodes are ready: the root of $\mathcal{T}$ is in the initial state of the controller of $\mathcal{N}$; the internal nodes of the successful rays are in the initial states of the processes of $\mathcal{N}$, and all clocks have values equal to 0.

Simulating Discrete Transitions. Below, we show how $\mathcal{T}$ simulates a rule of the form $[q_0 \rightarrow q'_0]\ [q_1; g_1 \rightarrow R_1; q'_1]\ \ \cdots\ [q_n; g_n \rightarrow R_n; q'_n]$. The root of $\mathcal{T}$ is in the state $\mathcal{T}(q_0)$. First, the root resets its clock to 0 (this is done so that it can later make sure that the simulation of the rule has not taken time). The simulation consists of different phases, where in each phase the root tries to identify a ray that can play the role of process k for $1 \leq k \leq n$. To find the first ray, it sends a broadcast message. An (internal) node that receives the broadcast and whose local state is q_1 may either decide to ignore the message or to try to become the node that simulates the first process in the rule. In the latter case it will enter a temporary state from which it initiates a sub-protocol whose goal is to confirm its status as the simulator of the first process. In doing so, the node has guessed (perhaps wrongly) that its clocks satisfy the values specified by the guard. If the node has guessed wrongly it will eventually be excluded from the rest of the simulation (will remain passive in the rest of the simulation). At the end of this phase, exactly one node will be chosen among the ones that have correctly guessed that their clocks satisfy g_1. The successful node will be the one that plays the role of the first process. The sub-protocol proceeds as follows: (i) The internal node checks whether the value of its clock satisfies the guard g_1. Recall that each node contains one clock. Since the guard g_1 only compares the clocks x_1, x_2 with constants, the conditions of g_1 can be tested on each of x_1 and x_2 separately. If the clock of the node does not satisfy g_1 (which means that x_1 does not satisfy g_1), the node will remain passive from now on (it has made the wrong guess). Otherwise, the node resets its clock if R_1 contains x_1, and then broadcasts a message (such a message is received by the leaf of the ray); (ii) The leaf checks whether the value of its clock satisfies the guard g_1 (i.e., if x_2 satisfies g_1); if *yes* it resets its clock if x_2 is included in R_1, and then broadcasts an acknowledgement. (iii) Upon receiving the above acknowledgement, the internal node declares itself ready for the next step by broadcasting an acknowledgement itself. At the same time, it moves to new local state and waits for a last acknowledgement from the root (described below) after which it will move to local state $\mathcal{T}(q'_1)$. (iv) When the root receives the acknowledgement it sends a broadcast declaring that it has successfully found a ray to simulate the first process. All the nodes in temporary states will now enter local states from which they remain passive. To prevent multiple nodes to play the role of the first process, the root enters en error state if it happens to receive acknowledgements from several internal nodes. The root now proceeds to identify the ray to simulate the second process. This continues until all n processes have been identified. Then the root makes one final move where the following events take place: (i) It moves its local state to $\mathcal{T}(q'_0)$ (ii) It sends a final boraodcast where the node ready for simulating the i^{th} process will now move to $\mathcal{T}(q'_i)$ for all $i : 1 \leq i \leq n$ (notice that there is at most one such node for each i). (iii) It checks that its clock is equal to 0 (the simulation of the rule has not taken any time).

Simulating Timed Transitions. This is done in a straightforward manner by letting time pass in $\mathcal{T}$ by the same amount as it has done in $\mathcal{N}$.

Cliques. We show that the reachability problem for the clique topology is undecidable if the nodes have two clocks.

Theorem 4. $\texttt{TAHN}-\texttt{Reach}(\texttt{CLIQUE}, 2)$ *is undecidable.*

We will build a protocol P with $\texttt{CLOCKS}(P) = 2$ which will simulate $\mathcal{N}$ on the clique topology. In a similar manner to the case of star topologies, the simulation consists of two phases.

Initialization Phase. The purpose of the initialization phase is to choose a node that will simulate the controller. This choice is done non-deterministically through a protocol that is initialized by a broadcast message. Notice that this protocol exists in all the nodes since they run the same pre-defined protocol. The first node which will perform the broadcast will become the controller (from now on we refer to this node as the *controller node*). When the controller node performs the above broadcast it moves to the state $\mathcal{T}(q_{ctrl}^{init})$, while all the other nodes will move to $\mathcal{T}(q_{proc}^{init})$.

Simulating Discrete Transitions. Below, we show how a rule of the form of the previous sub-section is simulated. In a similar manner to the case of stars, the controller node first resets its clock to 0. The simulation again consists of different phases, where in each phase the controller node tries to identify a ray that can play the role of process i for $1 \leq i \leq n$. To find the first process it sends a broadcast. A node that receives the broadcast, whose local state is q_1, and whose clocks (x_1 and x_2) satisfy the guard g_1, may decide to ignore the message or to try to become the node that simulates the first process in the rule. In the latter case, the node declares itself ready for the next step by broadcasting an acknowledgement. At the same time, it moves to a new local state and waits for a last acknowledgement from the controller node (described below) after which it will move to local state $\mathcal{T}(q_1')$. To prevent multiple nodes to play the role of the first process, the controller node enters en error state if it happens to receive acknowledgements from several nodes. The controller node now proceeds to identify the node to simulate the second process. This continues until all n processes have been identified. Then the controller node performs the same three steps as the ones in the final phase of the simulation described above for stars.

Bounded Path Topologies. Using the result of Theorem 3 we can show that the reachability problem can be extended to bounded path topologies. The result uses a reduction to the two-star case, thus we need to consider topologies in which the (number of vertices) simple paths can have 5 vertices in order to be able to rebuild stars with rays of depth 2.

For such a reduction, we need a preliminary protocol that discovers a two-star topology in an arbitrary graph in paths are allowed to have (at least) five nodes. The discovery protocol first selects root, internal and leaf candidates and then verify that they are connected in the desired way by sending all other nodes in their vicinities to a special null state. Combining the discovery protocol and the undecidability results for two-star topology we obtain the following theorem.

Theorem 5. $\texttt{TAHN}-\texttt{Reach}(\texttt{BOUNDED}(5), 1)$ *is undecidable.*

6 Decidability with Dense Time

In the previous section we showed that $\texttt{TAHN}-\texttt{Reach}(\texttt{STAR}(2), 1)$ is undecidable. In this section we consider two other classes of topologies for which reachability becomes decidable when nodes have a single clock, namely the class $\texttt{STAR}(1)$ and $\texttt{CLIQUE}$. A convenient way to prove these results is to resort to an extension of Timed Networks with transfers for which control state reachability is still decidable. When executed together with a rendez-vous, a transfer action from q to q' forces all processes in state q to move to state q' (as in transfer arcs for Petri nets). We give next a more detailed definition of the extended TN model.

Timed Networks with Transfers (TNT). A rule of a *timed network with transfer* (TNT) $\mathcal{N}$ combines rendez-vous synchronization and transfer actions. Namely, a TNT rule has the following form:

$$\begin{aligned} &[q_0 \to q_0'] \; [q_1; g_1 \to R_1; q_1'] \;\; \cdots \;\; [q_n; g_n \to R_n; q_n'] \\ &\qquad\qquad\quad [p_1; g_1' \to_t R_1'; p_1'] \cdots \; [p_\ell; g_\ell' \to_t R_\ell'; p_\ell'] \end{aligned}$$

where actions with $\to$ denote rendez-vous communication, whereas actions with $\to_t$ denote transfers. For $i : 1 \leq i \leq \ell$, $g_i \to_t R_i$ is a guarded transfer where g_i is a guard, R_i is a reset, $p_i, p_i' \in Q^{proc}$. We assume that there are no $i : 1 \leq i \leq \ell$ and $j : 1 \leq j \leq n$ such that $p_i = q_j$; and assume that $p_i \neq p_j$ if $i \neq j$. A TNT rule is enabled if the state of the controller is q_0 and if there are n processes with states $q_1, \cdots, q_n$ whose clock values satisfy the corresponding guards. The rule is executed by simultaneously changing the state of the controller to q_0' and together with the states of selected processes $p_1, \cdots, p_\ell$. Furthermore, each process in state p_i for $i : 1 \leq i \leq \ell$ whose clock value satisfies the guard g_i' moves to p_i' while its clocks are reset according to R_i'. Note that for a rule to be enabled there is no requirement on the presence or absence of processes in states $p_1, \cdots, p_\ell$.

The (controller state) reachability problem $\texttt{TNT}-\texttt{Reach}(K)$ for processes with K clocks is defined by replacing the TN transition relation with that for TNT. The following result then holds.

Theorem 6. $\texttt{TNT}-\texttt{Reach}(1)$ *is decidable.*

Sketch of proof. In [6], it is proved that $\texttt{TN}-\texttt{Reach}(1)$ is decidable. The proof is based on the general results in [1] based on a well quasi orderings of TN configurations under which the transition relation is monotonic. Monotonicity of the transition relation of a TNT still holds for the same ordering used in [6]. Furthermore, the algorithm used for computing predecessors can be extended in order to cope with transfer action. This extension is similar to that used for Data nets [12] or, for processes without clocks, to that used for Petri nets with transfer arcs. We prove next that in TAHNs with one clock restricted to the clique topology, the reachability problem is decidable.

Theorem 7. $\mathtt{TAHN}{-}\mathtt{Reach}(\mathtt{CLIQUE}, 1)$ *is decidable.*

Proof. We reduce $\mathtt{TAHN}{-}\mathtt{Reach}(\mathtt{CLIQUE}, 1)$ to $\mathtt{TNT}{-}\mathtt{Reach}(1)$. The reduction works as follows. Since in a clique graph all nodes are connected to each other, a broadcast message sent by a node is always received by all other nodes. In other words working with a clique is like working with a multiset of processes as in a TNT. Broadcast communication can then be simulated by using TNT rules in which the sender perfoms an individual step and reception of a message is simulated by transfer actions, one for each state in which the message can be received. Furthermore, we can insert a special rule to synchronize the controller with the local state we want to reach in the TAHN. Local state reachability corresponds then to control state reachability in the resulting TNT. The claim then follows from Theorem 6.

A similar positive result can be obtained for TAHN with 1 clock restricted to the star topology of depth 1. The main difference compared to the previous result is that in a star of depth 1 we have to distinguish the root (the central node) from the leaves. When the root performs a broadcast, it is transmitted to all the leaf nodes, but when a leaf performs one, only the root can receive it.

Theorem 8. $\mathtt{TAHN}{-}\mathtt{Reach}(\mathtt{STAR}(1), 1)$ *is decidable.*

Proof. We reduce $\mathtt{TAHN}{-}\mathtt{Reach}(\mathtt{STAR}(1), 1)$ to $\mathtt{TNT}{-}\mathtt{Reach}(1)$. Let $\mathcal{T}$ be a TAHN with star topology of depth 1. We construct a TNT N that simulates $\mathcal{T}$ as follows. Initially all TNT processes are in state q^i. We first define an initialization step in which the controller non-deterministically selects one of the processes in state q^i and elects it as the root of the simulated TAHN. All the remaining processes in state q^i are then used to simulate the leaves of the star. After this step, the simulation of each TAHN rule r is split in two TNT rules: one for the root process and one for a leaf process. A local rule is simulated by a rendez-vous (with no transfer) for the root process and by a rendez-vous for a leaf process. A broadcast rule executed by the root node is simulated with a TNT transfer action involving the root process and all the leaf processes, whereas a broadcast executed by a leaf is translated into a rendez-vous between a leaf and the root process only. As for cliques, we can add rules for the controller to observe when the root of a leaf process has reached a given local state. The claim then follows from Theorem 6.

7 Decidability with Discrete Time

In this section we study the reachability problem for Discrete Time Ad Hoc Networks (DTAHN). In this model clocks range over the natural numbers instead of the reals. Let μ' be the maximum constant used in the protocol rules and let $\mu = \mu' + 1$. When using discrete time, it is enough to restrict the valuation of clocks to the finite range $I = [0, \mu]$. This follows from the fact that guards of the form $x > c$ remain enabled when time passes once the clock associated to variable x reaches a value greater or equal to μ; while guards of the form $x \leq c$ remain disabled. Therefore, beyond μ we need not distinguish between different values for the same clock (see e.g. [3]). For every DTAHN $\mathcal{T}$, we can then define a finite-state process that describes the behavior of a node. We use $\mathcal{C}_{\mu,K}$ to denote configurations over undirected graphs with labels in $Q \times I^K$, where Q is the set of local states of a process, I is the interval $[0, \mu]$, and K is the number of clocks in each process.

The transition relation $\rightharpoonup_{\mathcal{T}}$ is obtained from that of TAHN by assuming that the valuation of clock variables is a function $\mathcal{X} : V \mapsto [X \rightarrow I]$ and by replacing the time step by the discrete time step defined as follows.
Discrete Time Step For configurations $\gamma = (\mathcal{Q}, \mathcal{X})$ and $\gamma' = (\mathcal{Q}', \mathcal{X}')$, we write $\gamma \rightharpoonup_{\mathcal{T}} \gamma'$ if, for all $v \in V$ and $x \in X$, the following conditions are satisfied:

- $\mathcal{Q}(\gamma') = \mathcal{Q}(\gamma)$,
- $\mathcal{X}'(v)(x) = \mathcal{X}(v)(x) + 1$, if $\mathcal{X}(v)(x) < \mu$
- $\mathcal{X}'(v)(x) = \mathcal{X}(v)(x) = \mu$, otherwise.

For a topology class *Top* and $K \geq 0$, the control state reachability problem `DTAHN−Reach`(*Top*, K) is the natural reformulation of the one defined for TAHN.

We show next that reachability is decidable when restricting the topology to the class of bounded path graphs `BOUNDED`(N) for any fixed $N > 1$. The decision procedure is obtained by resorting to the theory of well-structured transition systems [1]. The procedure is based on a symbolic backward exploration algorithm in which we use *constraints* to finitely represent sets of configurations of *variable size* taken from the class `BOUNDED`(N) (the configurations may potentially belong to different TAHNs).

Ordering. We first introduce the following ordering between configurations of variable size. Given configurations $\gamma = (\mathcal{Q}, \mathcal{X})$ defined over $G = (V, E)$ and $\gamma' = (\mathcal{Q}', \mathcal{X}')$ defined over $G' = (V', E')$, $\gamma \preceq \gamma'$ iff there exists an injective function $h : V \mapsto V'$ such that:

- $\forall u, u' \in V$, $(u, u') \in E$ if and only if $(h(u), h(u')) \in E'$;
- $\forall u \in V$, $\mathcal{Q}(u) = \mathcal{Q}'(h(u))$ and $\mathcal{X}(u) = \mathcal{X}'(h(u))$.

An upward closed set U satisfies the property that $U = \{\gamma' \mid \gamma \preceq \gamma',\ \gamma \in U\}$. The following property then holds.

Proposition 1. *If U is an upward closed set of configurations (of variable size), then $Pre(U) = \{\gamma \mid \gamma \rightharpoonup_{\mathcal{T}} \gamma',\ \gamma' \in U\}$ is still upward closed wrt. $\preceq$.*

Proof. In [7] it has been proven, that for untimed AHN, selective broadcast is monotone w.r.t. $\preceq$. We observe that DTAHN restricted to configurations in $\mathcal{C}_{\mu,K}$ can be viewed as untimed AHN extended with a time step transition. Thus, we just have to show monotonicity wrt. a time step $\gamma \rightharpoonup_{\mathcal{T}} \gamma'$. Since time can always proceed, for every $\beta \succeq \gamma$ there is a configuration β' such that $\beta \rightharpoonup_{\mathcal{T}} \beta'$ with a time step. Now since h is label preserving both time steps will have the same effect on nodes identified by h and thus $\gamma' \preceq \beta'$.

Constraints. Assume a given process definition P. A constraint is a tuple $\Phi = (G, \mathcal{Q}, \mathcal{X})$, where $G = (V, E)$ is a graph in $\mathtt{BOUNDED}(N)$, and $\mathcal{Q}$ and $\mathcal{X}$ are defined on the set of vertices V. We call γ_Φ the associated configuration $(\mathcal{Q}, \mathcal{X})$ over the graph G. The denotation of a constraint Φ is defined as the infinite set of configurations $[\![\Phi]\!] = \{\gamma' \mid \gamma_\Phi \preceq \gamma'\}$ with variable size and topology. The following proposition then holds.

Proposition 2. *Given a constraint Φ, there exists an algorithm to compute a finite set of constraints whose denotation corresponds to $Pre([\![\Phi]\!])$.*

Proof. We can extend the symbolic predecessor operator defined for untimed AHN in [7] by adding some new conditions and steps. In order to correctly obtain predecessors of γ, rules of the form $\left(q, g \xrightarrow{e} R, q'\right)$ must be applied only to nodes in state q' which satisfy the post-condition R. When R is not empty, i.e. some clocks have been reset, we do not know their values before the step. In this case it is necessary to add a predecessor to the set $Pre(\gamma)$ for every possible combination of $i \in I$ for those clocks – keeping the old values of the node from configuration γ otherwise.

As an addition to regular protocol rules, we also have to consider time steps. Thus, for every configuration γ that does not have 0-valued clocks, we have to add to its set of predecessors every configuration γ' such that γ' can make a time step into γ.

Theorem 9. $\mathtt{DTAHN-Reach}(\mathtt{BOUNDED}(N), K)$ *is decidable for any $N \geq 1$ and $K \geq 0$.*

Proof. From propositions 1 and 2, we can apply the general results in [1] to define a backward search algorithm working on upward closed sets of extended configurations represented by their finite basis. The finite set of constraints (G, Q, X) in which G is a single node v, $Q(v) = q$, and $X(v) \in I$ can be used as finite representation of all configurations containing the control state q. Furthermore, for a fixed $N \geq 1$ the induced subgraph relation is a well-quasi-ordering on the class of N-bounded path graphs [10]. This implies that for any sequence of constraints $\Phi_0\Phi_1 \ldots$ there exists i and j with $i < j$ s.t. $\Phi_i \preceq \Phi_j$. This property ensures the termination of the symbolic backward reachability algorithm.

8 Conclusions

We have studied local state reachability for Timed Ad Hoc Networks in different classes of topologies by taking the number of clocks of each node as a parameter. In order to refine our analysis, it could be interesting to investigate restricted form of guards, e.g., open and closed specifications as in [5], to obtain decidable fragments of the full model. Furthermore, similarly to the untimed case [9], it would be interesting to evaluate the impact of failures and delays in the decidability of local reachability for TAHNs. From a more practical point of view, by exploiting symbolic representations of timed regions it could be interesting to apply backward reachability analysis in order to extract preconditions (in form of minimal representation of topologies) for violations of safety conditions in protocol models like those considered in [11]. Finally, finding abstractions for topology- and time-dependent broadcast protocols that could be used to reason on arbitrary configurations is another interesting direction for future work.

References

1. Abdulla, P., Cerans, K., Jonsson, B., Tsay, Y.: General decidability theorems for infinite-state systems. In: LICS 1996, pp. 313–321. IEEE Computer Society, Los Alamitos (1996)
2. Abdulla, P., Delzanno, G., Rezine, O., Sangnier, A., Traverso, R.: On the verification of timed ad hoc networks. Disi-11-05. Univ. of Genova (July 2011)
3. Abdulla, P., Deneux, J., Mahata, P.: Multi-clock timed networks. In: LICS 2004, pp. 345–354. IEEE Computer Society, Los Alamitos (2004)
4. Abdulla, P., Nylén, A.: Timed petri nets and bqos. In: Colom, J.-M., Koutny, M. (eds.) ICATPN 2001. LNCS, vol. 2075, pp. 53–70. Springer, Heidelberg (2001)
5. Abdulla, P.A., Deneux, J., Mahata, P.: Closed, open, and robust timed networks. ENTCS 138(3), 117–151 (2005)
6. Abdulla, P.A., Jonsson, B.: Model checking of systems with many identical timed processes. TCS 290(1), 241–264 (2003)
7. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 313–327. Springer, Heidelberg (2010)
8. Delzanno, G., Sangnier, A., Zavattaro, G.: On the power of cliques in the parameterized verification of ad hoc networks. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 441–455. Springer, Heidelberg (2011)
9. Delzanno, G., Sangnier, A., Zavattaro, G.: Verification of ad hoc networks with node and communication failures. Disi-11-06. Univ. of Genova (July 2011)
10. Ding, G.: Subgraphs and well quasi ordering. J. of Graph Theory 16(5), 489–502 (1992)
11. Fehnker, A., van Hoesel, L., Mader, A.: Modelling and verification of the LMAC protocol for wireless sensor networks. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 253–272. Springer, Heidelberg (2007)
12. Lazic, R., Newcomb, T., Ouaknine, J., Roscoe, A.W., Worrell, J.: Nets with tokens which carry data. Fund. Inf. 88(3), 251–274 (2008)
13. Merro, M., Ballardin, F.F., Sibilio, E.: A timed calculus for wireless systems. In: Arbab, F., Sirjani, M. (eds.) FSEN 2009. LNCS, vol. 5961, pp. 228–243. Springer, Heidelberg (2010)
14. Nanz, S., Nielson, F., Nielson, H.: Static analysis of topology-dependent broadcast networks. Inf. Comput. 208(2), 117–139 (2010)
15. Saksena, M., Wibling, O., Jonsson, B.: Graph grammar modeling and verification of Ad Hoc Routing Protocols. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 18–32. Springer, Heidelberg (2008)
16. Singh, A., Ramakrishnan, C.R., Smolka, S.A.: Query-based model checking of ad hoc network protocols. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 603–619. Springer, Heidelberg (2009)
17. The uppaal web page, http://www.uppaal.com/

Incremental Computation of Succinct Abstractions for Hybrid Systems[*]

Tomáš Dzetkulič and Stefan Ratschan

Institute of Computer Science, Academy of Sciences of the Czech Republic

Abstract. In this paper, we introduce a new approach to computing abstractions for hybrid dynamical systems whose continuous behavior is governed by non-linear ordinary differential equations. The abstractions try to capture the reachability information relevant for a given safety property as succinctly as possible. This is achieved by an incremental refinement of the abstractions, simultaneously trying to avoid increases in their size as much as possible. The approach is independent of a concrete technique for computing reachability information, and can hence be combined with whatever technique suitable for the problem class at hand. We illustrate the usefulness of the technique with computational experiments.

1 Introduction

In this paper, we study hybrid (dynamical) systems, that is, systems with both discrete and continuous state and evolution. We address the computation of abstractions of hybrid systems that capture a given safety property as succinctly as possible. In other words, given a hybrid system, a set of initial states, and a set of states considered to be unsafe, we try to characterize the set of trajectories (of unbounded length) from an initial to an unsafe state. In cases where the input system does not have such a trajectory (i.e., it fulfills the safety property), we aim at computing an empty abstraction, which verifies the safety property at hand. In other cases, the computed abstraction can be used to guide testing/falsification of the input system.

The abstractions are computed incrementally. This has the obvious advantages of such incremental computation, which are analogous to the case of discrete systems. However, in addition to that, for systems with (complex) continuous dynamics, incrementality has another important aspect: In such cases, even over bounded time, exact reachability computation is possible only for very special cases, and hence (unlike for discrete systems) one *has to* use over-approximation. It is a-priori not clear, how much and where to over-approximate in order to prove a given property. By incremental computation one can adapt the level of over-approximation to the given problem.

[*] This work was supported by Czech Science Foundation grants 201/08/J020 and 201/09/H057, MŠMT project number OC10048 and institutional research plan AV0Z100300504.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 271–285, 2011.

Our technique is parametric in the method used for computing reachability information. We have an implementation that is based on hyper-rectangles and interval constraint propagation [4]. But instantiations with various alternative techniques are possible.

Moreover, our technique is orthogonal to other techniques for abstraction refinement, especially counter-example guided abstraction refinement (CEGAR) [1,7]: The essential step of our technique tries to capture information about the given safety verification problem *without* increasing the size of the abstraction. Moreover, this step is not inherently exponential in the problem dimension, hence allowing the computation of abstractions of high dimension. In contrast to that, CEGAR approaches are inherently based on increasing the number of states of the abstraction. Both approaches can be used in combination. See also the approach of abstraction slicing [6] in a discrete CEGAR setting.

Computational experiments show the usefulness of the approach.

Concerning other related work, some approaches to hybrid systems verification do exploit incrementality in reachability computation [12,11]. But, reuse of analyses only concerns dropping initial/unsafe states that have been shown not to lie on any error trajectory—no reuse is done concerning the analysis itself.

Parts of the material in this paper can be viewed as an adaptation and extension of the framework of abstract interpretation [8,13] to a situation where abstractions are incrementally refined, and where specific properties of continuous time systems are exploited. The reader will find more discussion on the relation to that approach throughout the paper.

Our previous approach for hybrid systems verification [16] computes abstractions based on an algorithm that is a special case of the method presented here. However, as our computational experiments will show, variants of our method that are different from that special case, show much better behavior, especially for hybrid systems with cyclic behavior. Moreover, our previous approach was restricted to boxes, and a very specific way of computing reachability information based on the mean-value theorem and interval constraint propagation.

The content of the paper is as follows: in Section 2 we describe the used formalism for modelling hybrid systems, in Section 3 we describe how to incrementally improve an abstraction of a given hybrid system, in Section 4 we show how to further improve the technique, in Section 5 we discuss a concrete implementation of the method, in Section 6 we discuss the behavior of this implementation on some computational experiments, and in Section 7 we conclude the paper.

2 Hybrid Systems

In this section, we briefly recall our formalism for modelling hybrid systems. It captures many relevant classes of hybrid systems, and many other formalisms for hybrid systems in the literature are special cases of it. We use a set S to denote the discrete modes of a hybrid system, where S is finite and nonempty. $I_1, \ldots, I_k \subseteq \mathbb{R}$ are compact intervals over which the continuous variables of a hybrid system range. Φ denotes the state space of a hybrid system, i.e., $\Phi = S \times I_1 \times \cdots \times I_k$.

Definition 1. *A* hybrid system *H is a tuple* $(\texttt{Flow}, \texttt{Jump}, \texttt{Init}, \texttt{Unsafe})$*, where* $\texttt{Flow} \subseteq \Phi \times \mathbb{R}^k$*,* $\texttt{Jump} \subseteq \Phi \times \Phi$*,* $\texttt{Init} \subseteq \Phi$*, and* $\texttt{Unsafe} \subseteq \Phi$*.*

Informally speaking, the predicate `Init` specifies the initial states of a hybrid system and `Unsafe` the set of unsafe states that should not be reachable from an initial state. The relation `Flow` specifies the possible continuous flow of the system by relating states with corresponding derivatives, and `Jump` specifies the possible discontinuous jumps by relating each state to a successor state. Formally, the behavior of H is defined as follows:

Definition 2. *A* flow *of length $l \geq 0$ in a mode $s \in S$ is a function $r : [0, l] \rightarrow \Phi$ such that the projection of r to its continuous part is differentiable and for all $t \in [0, l]$, the mode of $r(t)$ is s. A* trajectory *of H is a sequence of flows $r_0, \ldots, r_p$ of lengths $l_0, \ldots, l_p$ such that for all $i \in \{0, \ldots, p\}$,*

1. *if $i > 0$ then $(r_{i-1}(l_{i-1}), r_i(0)) \in \texttt{Jump}$, and*
2. *if $l_i > 0$ then $(r_i(t), \dot{r}_i(t)) \in \texttt{Flow}$, for all $t \in [0, l_i]$, where $\dot{r}_i$ is the derivative of the projection of r_i to its continuous component.*

A (concrete) error trajectory *of a hybrid system H is a trajectory $r_0, \ldots, r_p$ of H such that $r_0(0) \in \texttt{Init}$ and $r_p(l) \in \texttt{Unsafe}$, where l is the length of r_p. H is* safe *if it does not have an error trajectory.*

In the rest of the paper we will assume an arbitrary, but fixed hybrid system H. We will denote the set of its error trajectories by $\mathcal{E}$.

Instead of defining some concrete syntax in which hybrid systems are described, we keep this paper independent of concrete syntax, and require the user to provide certain operations on hybrid systems that we will introduce in the next section.

3 Incremental Abstract Forward/Backward Computation

For hybrid systems with complex continuous dynamics even bounded time reach set computation necessarily involves over-approximation. In such cases one would like to first compute approximate information using loose over-approximation, and then incrementally refine this.

Our approach will be based on an incremental refinement of a covering of the hybrid systems state space by connected sets that we will call *regions*. We will form the regions in such a way that no two regions will overlap (i.e., regions are allowed to intersect, but only on their boundaries). The method is independent of the class of regions used. In the instantiation of the method used by our implementation (see Section 5), the regions will be formed by pairs consisting of a mode and a Cartesian product of intervals (i.e., a *box*), but other classes of regions (e.g., based on polyhedra) are equally conceivable.

Definition 3. *An* abstraction *is a graph whose vertices (which we call* abstract states*) may be labelled with labels* `Init` *or* `Unsafe`*. Moreover, to each abstract state, we assign a region. We call the edges of an abstraction* abstract transitions*.*

By abuse of notation, we will usually use the same notation for an abstract state and the region assigned to it.

We call a sequence of abstract states $a_1, \dots, a_n$ an *abstract trajectory.* If all abstract states and all transitions between successive abstract states in an abstract trajectory belong to an abstraction $\mathcal{A}$, we call it an $\mathcal{A}$-*abstract trajectory* and we denote it by $a_1 \to \cdots \to a_n$. An ($\mathcal{A}$-)abstract trajectory represents the set of concrete trajectories that begin in the region of a_1, move from one abstract state region to the next only if there is a corresponding concrete transition, and end in the region of a_n. We denote this set by $[\![a_1, \dots, a_n]\!]$ for a given abstract trajectory or $[\![a_1 \to \cdots \to a_n]\!]$ for some $\mathcal{A}$-abstract trajectory.

This can be formalized as follows: We define a *splitting* of a flow l to be a sequence of flows $s_1, \dots, s_j$ such that $s_1(0) = l(0)$, $s_j(\mathit{length}(s_j)) = l(\mathit{length}(l))$ and if $i > 1$ then $s_{i-1}(\mathit{length}(s_{i-1})) = s_i(0)$. A *trajectory splitting* is a concatenation of splittings of its individual contained flows. $[\![a_1, \dots, a_n]\!]$ then is the set of all concrete trajectories $r_1, \dots, r_p$ that have a trajectory splitting $q_1, \dots q_n$, such that for all i, the mode of abstract state a_i is the same as the projection of q_i to its discrete part and such that the projection of q_i to its continuous part is in the region of a_i.

An $\mathcal{A}$-*abstract error trajectory* is an $\mathcal{A}$-abstract trajectory $a_1 \to \cdots \to a_n$ such that in $\mathcal{A}$, a_1 is labelled initial, and a_n is labelled unsafe.

An abstraction $\mathcal{A}$ represents the set of all concrete trajectories $[\![a_1 \to \cdots \to a_n]\!]$ for abstract error trajectories $a_1 \to \cdots \to a_n$ in the abstraction $\mathcal{A}$. We denote this set by $[\![\mathcal{A}]\!]$.

The intuition is that, during abstraction refinement, the abstraction stays an over-approximation of the set of error trajectories $\mathcal{E}$ of a given system. We say that an abstraction $\mathcal{A}^*$ is a *refinement* of an abstraction $\mathcal{A}$ iff

- the abstraction $\mathcal{A}^*$ represents less trajectories than $\mathcal{A}$, that is, $[\![\mathcal{A}^*]\!] \subseteq [\![\mathcal{A}]\!]$, and
- the abstraction $\mathcal{A}^*$ does not lose error trajectories that are present in $\mathcal{A}$, that is $[\![\mathcal{A}^*]\!] \supseteq [\![\mathcal{A}]\!] \cap \mathcal{E}$.

Now we will come up with an algorithm that will incrementally improve an abstraction by refining it, *without* increasing the number of abstract states in the abstraction. Note that, in particular, $\mathcal{A}$ is a refinement of $\mathcal{A}$ itself, but in practice we will try to remove as many trajectories from the abstraction as possible.

Given abstract states a and a', we will assume a procedure $\mathit{InitReach}(a)$ that computes an over-approximation of the set of points in a that are reachable from an initial point in a, and a procedure $\mathit{Reach}(a, a')$ that computes an over-approximation of the set of points in a' reachable from a according to the system dynamics (here we do not assume any time bound, implementations of those procedures that compute reachability over bounded time would only require slight modifications of our algorithms). Our method is independent of the concrete technique used to compute those procedures. Still, in Section 5.2 we will present an example of how this can be implemented in practice. We assume that smaller inputs improve the precision of these operations, that is:

- $a_1 \subseteq a_2$ implies $\mathit{InitReach}(a_1) \subseteq \mathit{InitReach}(a_2)$
- $a_1 \subseteq a_2$ and $a_1' \subseteq a_2'$ implies $\mathit{Reach}(a_1, a_1') \subseteq \mathit{Reach}(a_2, a_2')$

Furthermore, we assume that these procedures exploit information about empty inputs, that is:

- $a = \emptyset$ implies $\mathit{InitReach}(a) = \emptyset$
- $a = \emptyset$ implies $\mathit{Reach}(a, a') = \emptyset$
- $a' = \emptyset$ implies $\mathit{Reach}(a, a') = \emptyset$

In the following, we require the existence of operations $\sqsubseteq$ and $\uplus$ on regions, with the following properties.

- $\sqsubseteq$ such that if $a^* \sqsubseteq a$, then for all $n \in \mathbb{N}$, for all $i \in \{1 \dots n\}$ and for all regions $b_1 \dots b_{i-1}, b_{i+1} \dots b_n$ we have that $[\![b_1, \dots, b_{i-1}, a^*, b_{i+1}, \dots, b_n]\!] \subseteq [\![b_1, \dots, b_{i-1}, a, b_{i+1}, \dots, b_n]\!]$ i.e., less concrete trajectories follow a given abstract trajectory after replacing an abstract state by smaller one wrt. $\sqsubseteq$ operation.
- $\uplus$ s.t. for all regions $a_1, a_2, b : a_1 \sqsubseteq b \wedge a_2 \sqsubseteq b$ implies $a_1 \uplus_b a_2 \sqsubseteq b$, $a_1 \sqsubseteq a_1 \uplus_b a_2$ and $a_2 \sqsubseteq a_1 \uplus_b a_2$.

Since in our case abstract states represent sets, this can be ensured by the following:

- $\uplus$ s.t. for all $a_1, a_2 \subseteq b$: $a_1 \cup a_2 \ \subseteq \ a_1 \uplus_b a_2$ and $a_1 \uplus_b a_2 \ \subseteq \ b$
- $\sqsubseteq$ s.t. $a_1 \sqsubseteq a_2$ iff $a_1 \subseteq a_2$

This is our natural interpretation of $\uplus$ and $\sqsubseteq$. However, different choices are possible, as long as they fulfill the above properties: For certain representations of regions it might be convenient to use a weaker form of $\sqsubseteq$, for efficiency reasons. Also, later (Section 4.1) we will see that for being able to encode more information into abstract states, different interpretations of those symbols are convenient.

In the instantiation of the method with boxes, $a_1 \uplus_b a_2$ is the smallest box that includes both argument boxes a_1 and a_2, but does not exceed b (i.e., box union intersected with bounding box), and $\sqsubseteq$ is the subset operation on boxes. Note that for $a_1, a_2 \subseteq \ b$ defining $a_1 \sqsubseteq a_2$ iff $a_1 \uplus_b a_2 = a_2$ fulfills the above property.

The following algorithm (which we will call *pruning algorithm*) computes a refinement of a given abstraction $\mathcal{A}$:

```
$\mathcal{A}^* \leftarrow$ copy of $\mathcal{A}$, all regions set to $\emptyset$, no initial labels, no edges
// from now on, for every abstract state $a$ of $\mathcal{A}$,
// we denote by $a^*$ the corresponding abstract state of $\mathcal{A}^*$
for all $a \in \mathcal{A}$, $a$ is initial
    $a^* \leftarrow \mathit{InitReach}(a)$
    if $a^* \neq \emptyset$ then mark $a^*$ as initial
while there is a pair of abstract states $(a_1, a_2)$ in $\mathcal{A}$ with
    $a_1 \rightarrow a_2$, s.t. $\mathit{Reach}(a_1^*, a_2) \not\sqsubseteq a_2^*$ or ($a_1^* \not\rightarrow a_2^*$ and $\mathit{Reach}(a_1^*, a_2) \neq \emptyset$) do
        if $a_1^* \not\rightarrow a_2^*$ in $\mathcal{A}^*$ then introduce an edge $a_1^* \rightarrow a_2^*$ into $\mathcal{A}^*$
        if $\mathit{Reach}(a_1^*, a_2) \not\sqsubseteq a_2^*$ then $a_2^* \leftarrow (a_2^* \uplus_{a_2} \mathit{Reach}(a_1^*, a_2))$
return $\mathcal{A}^*$
```

In contrast to similar algorithms in abstract interpretation, this algorithm exploits and refines the knowledge already available in the abstraction $\mathcal{A}$. In contrast to CEGAR approaches, the algorithm does *not* increase the size (i.e., the number of nodes) of the abstraction. Still it deduces some interesting information:

Theorem 1. *The result of the pruning algorithm is a refinement of the input abstraction $\mathcal{A}$.*

Proof. We have to prove two items:

- $[\![\mathcal{A}^*]\!] \subseteq [\![\mathcal{A}]\!]$: This follows from the following:
  - the set of initial/unsafe marks of $\mathcal{A}^*$ is a subset of the set of marks of $\mathcal{A}$
  - the set of edges of $\mathcal{A}^*$ is a subset of the set of edges of $\mathcal{A}$
  - the abstract states of $\mathcal{A}^*$ are subsets of the corresponding abstract states of $\mathcal{A}$ since $\mathit{InitReach}(a) \subseteq a$, and $a_2^* \uplus_{a_2} \mathit{Reach}(a_1^*, a_2) \subseteq a_2$ from the definition of $\uplus$.
- $[\![\mathcal{A}^*]\!] \supseteq [\![\mathcal{A}]\!] \cap \mathcal{E}$: Let T be an $\mathcal{A}$-error trajectory in $[\![\mathcal{A}]\!] \cap \mathcal{E}$. We prove that T is an element of $[\![\mathcal{A}^*]\!]$. Let $a_1 \rightarrow \cdots \rightarrow a_n$ be an $\mathcal{A}$-abstract error trajectory s.t. $T \in [\![a_1 \rightarrow \cdots \rightarrow a_n]\!]$. We prove that for the corresponding $\mathcal{A}^*$-abstract trajectory $a_1^* \rightarrow \cdots \rightarrow a_n^*$, also $T \in [\![a_1^* \rightarrow \cdots \rightarrow a_n^*]\!]$. Let T_i be a fragment of trajectory T before the transition $a_i \rightarrow a_{i+1}$ is made. We will prove by induction that for all i: $T_i \in [\![a_1^* \rightarrow \cdots \rightarrow a_i^*]\!]$ and observation $T = T_n$ concludes the proof.
  - a_1 contains the initial point of T and $\mathit{InitReach}(a_1)$ contains all points of T in a_1. After the first loop of the algorithm, a_1^* is initial in $\mathcal{A}^*$ and it is equal to $\mathit{InitReach}(a_1)$. After that, all regions in $\mathcal{A}^*$ only increase w.r.t. $\sqsubseteq$ operation and that implies that $T_i \in [\![a_1^*]\!]$.
  - We assume that for some $i < n$: $T_i \in [\![a_1^* \rightarrow \cdots \rightarrow a_i^*]\!]$. Since T leaves a_i to a_{i+1}, a_i^* contains all the points of T in a_i, $\mathit{Reach}(a_i^*, a_{i+1})$ contains all the points of T in a_{i+1}. Since $\mathit{Reach}(a_i^*, a_{i+1})$ is non-empty, the abstract transition $a_i^* \rightarrow a_{i+1}^*$ is introduced into $\mathcal{A}^*$ and from the while cycle termination condition, we have: $\mathit{Reach}(a_i^*, a_{i+1}) \sqsubseteq a_{i+1}^*$, thus $T_{i+1} \in [\![a_1^* \rightarrow \cdots \rightarrow a_{i+1}^*]\!]$.

□

Note that the second part of the proof does *not* depend on the definition of $\uplus$. Hence, for ensuring that no error trajectory is lost by the algorithm, $\uplus$ does not necessarily have to fulfill the requirement stated above. This requirement just ensures that all computed reachability information is captured, and hence will not have to be re-computed again.

Note moreover, that it is a-priori not clear, that the pruning algorithm terminates. However, termination can be ensured, for example, by using a representation for which, for given regions a and b, there is not infinite chain $a \sqsubseteq a_1 \sqsubseteq a_2 \sqsubseteq \cdots \sqsubseteq b$. See Section 5.1 for a more detailed discussion of this issue.

As already mentioned, the pruning algorithm tries to deduce information about a given system without increasing the size of the abstraction. In cases, where it can deduce no more information, we have to fall back to some increase of the size of the abstraction (cf. to a similar approach in constraint programming where one falls back to exponential-time splitting, when polynomial-time deduction does not succeed any more).

We do this by the *Split* operation that chooses an abstract state and splits it into two, copying all the involved edges and introducing edges between the two new states. All the labels and abstract transitions to other abstract states are copied as well. Moreover, two new abstract transitions that connect the original abstract state with its copy are added. The region assigned to the abstract state is equally split among two abstract states. Such an refinement decreases the amount of over-approximation in subsequent calls to the pruning algorithm due to the properties of the *Reach* and *InitReach*. We chose such a simple splitting strategy to show the usefulness of our approach in isolation. However, it is possible to use much more sophisticated splitting strategies, for example, one could use one CEGAR step [1,7] instead of our simple splitting technique.

It is clear that the pruning algorithm can also be done backward in time (i.e., removing parts of the abstraction not leading to an unsafe state)[12,11]. We will denote the resulting algorithm by $Prune^{-}(\mathcal{A})$.

Now we have to following overall algorithm for computing increasingly fine abstractions:

```
initialize $\mathcal{A}$ with an arbitrary abstraction such that
    $[\![A]\!]$ contains all error trajectories of the input system
**while** there is an $\mathcal{A}$-abstract error trajectory
    $\mathcal{A} \leftarrow Prune(\mathcal{A})$
    $\mathcal{A} \leftarrow Prune^{-}(\mathcal{A})$
    $\mathcal{A} \leftarrow Split(\mathcal{A})$
**return** "safe"
```

The most simple way to initialize the abstraction $\mathcal{A}$ in this algorithm is to use the trivial abstraction containing just one vertex for every mode marked with both `Init` and `Unsafe` marks, containing a transition to all other vertices and itself, and a region containing the whole state space of the input system. However, one may also reflect some known structure of the input system, see Section 5 for more details.

Since neither pruning nor splitting removes an error trajectory, the absence of an $\mathcal{A}$-abstract error trajectory at the termination of the while loop implies the absence of an error trajectory of the original system. Hence, in such a case, the algorithm correctly returns the information that the input system was safe. In cases where the input does have an error trajectory, this algorithm does not terminate. However, in such cases, the algorithm maintains an abstraction that, at any time, can be used by other algorithms [17] for searching for this error trajectory.

Note that forward pruning may enable further backward pruning and vice versa, hence the algorithm may be extended in such a way that forward and backward pruning are done in a loop until no further improvement occurs. If either forward or backward pruning is dropped from the algorithm, it will incrementally compute a tighter and tighter over-approximation of the (forward/backward) reach set.

4 Improvements

In this section we introduce three improvements to the basic pruning algorithm. The first improvement exploits the specific structure of hybrid systems (continuous time, discontinuous jumps) and the two other improvements introduce more incrementality into the way the algorithms handles the abstraction.

4.1 Exit Regions

When computing $\mathit{Reach}(a_1^*, a_2)$ we get an over-approximation of the set of points in a_2 reachable from a_1^* according to the system dynamics. However, trajectories can leave a_1^* not arbitrarily, but only at points fulfilling certain conditions:

- flows can leave a_1^* only over its boundary, and
- jumps can leave a_1^* only over parts of a_1^* belonging to the projection of the set Jump to its first part corresponding to jump source, that is, over the set $\{(m, x) \in S \times \mathbb{R}^k \mid \exists x' \in \mathbb{R}^k, m' \in S \,.\, ((m, x), (m', x')) \in \texttt{Jump}\}$.

Hence, in the computation of $\mathit{Reach}(a_1^*, a_2)$, instead of the full region a_1^* we can use a subset fulfilling this condition. However, a_1^* already is an over-approximation. Hence, it is better to directly compute an over-approximation of the points fulfilling this condition, as follows:

We assume that the function $\mathit{InitReach}(a)$, in addition to an over-approximation of the set of points in a that are reachable from an initial point in a, also computes an over-approximation of the set of states in a through which a trajectory starting from an initial point in *a leaves a*. Moreover, we assume that $\mathit{Reach}(a, a')$ in addition to an over-approximation of the set of points in a' reachable from a, computes an over-approximation of the set of states in a' through which a trajectory coming from *a leaves a′*.

We store those additional regions with abstract states, calling them *exit regions*. For extending the operations $\uplus$ and $\sqsubseteq$ to abstract states consisting of a region and an exit region we will need the following:

Lemma 1. *Assume a region a that contains a trajectory T, an $a^* \sqsubseteq a$ that also contains T and an exit region a_e containing the point L where T leaves a. Then T also leaves a^* at L.*

Proof. Since a^* contains all points of T in a, it also contains L. Let us assume that T leaves a^* at L^* and that $L^* \neq L$. Since a^* contains T and both L and L^*, it also contains a flow r, that is a part of the trajectory T, such that $r(0) = L \wedge r(length(r)) = L^*$. Since $a^* \sqsubseteq a$, flow r is also in a and that contradicts the assumption that T leaves a at L. □

Hence, T leaves a^* at a point that already was an element of the exit region a_e of the original box a. This motivates us to extend the check $\sqsubseteq$ to abstract states with exit regions in such a way that for abstract states (i.e., region/exit region pairs) (c_r, c_e) and (d_r, d_e), $(c_r, c_e) \sqsubseteq (d_r, d_e)$ iff $c_r \sqsubseteq d_r$ and $c_e \sqsubseteq d_e$. Moreover, $\uplus$ will also apply the corresponding operation on both the region and exit region.

This makes it possible to implement *InitReach* and *Reach* in such a way that they are compatible with $\sqsubseteq$.

Now, we extend the semantics of abstract trajectories in such a way that concrete trajectories have to leave abstract states through their exit regions. The correctness of the pruning algorithm is preserved, if the new operations $\uplus$ and $\sqsubseteq$ still fulfill the necessary properties. This is clearly the case for $\uplus$. But also $\sqsubseteq$ again fulfills the required property:

Theorem 2. *For $\sqsubseteq$ extended with exit regions, for all abstract states $a = (a_r, a_e)$ and $a^* = (a_r^*, a_e^*)$, if $a_r^* \sqsubseteq a_r$ and $a_e^* \sqsubseteq a_e$ then $a^* \sqsubseteq a$.*

Proof. Let T be a trajectory in $[\![b_1, \ldots b_{i-1}, a^*, b_{i+1}, \ldots, b_n]\!]$. We show that this trajectory is also in $[\![b_1, \ldots, b_{i-1}, a, b_{i+1}, \ldots, b_n]\!]$. Since $a_r^* \sqsubseteq a_r$, T is covered by the region a_r. All we have to prove is that T leaves a at the point in a_e. From Lemma 1, we know that T leaves a and a^* at the same point L. Since T is in $[\![b_1, \ldots b_{i-1}, a^*, b_{i+1}, \ldots, b_n]\!]$, L belongs to a_e^* and since $a_e^* \sqsubseteq a_e$ it also belongs to a_e. □

Corollary 1. *The result of the pruning algorithm extended with exit regions is a refinement of the input abstraction $\mathcal{A}$.*

Exit regions can be computed in both forward (i.e., $Prune(\mathcal{A})$) and backward (i.e., $Prune^-(\mathcal{A})$) computation. We will call such a region computed during backward computation an *enter region.* Exit regions are computed only during forward computation, while enter regions are computed only during backward computation. When doing computation in one direction, the dual exit regions do not change.

Moreover, in the computation of $Reach(a_1^*, a_2)$, one can exploit not only the exit region of a_1^*, where the trajectory leaves a_1^*, but also the enter region of a_2, where the trajectory has to enter the region of a_2. Such an information can then be used by the underlying reachability computation algorithm to constrain the reachable states.

Furthermore, in $Split(\mathcal{A})$, new boundaries are created. A trajectory can now enter and exit the region through this new boundary and we have to create new abstract states in such a way that the new boundary is part of the enter and exit regions. For new abstract state a^*, new boundary b and original exit region a_e, we create a new exit region a_e^* in such a way that $(a_e \cap a^*) \cup b \subseteq a_e^*$. This clearly does not change the set of represented error trajectories.

4.2 Avoiding Redundant Edge Checks

One disadvantage of the pruning algorithm is that it may do redundant tests for the condition $\mathit{Reach}(a_1^*, a_2) \not\subseteq a_2^*$ in the update function. Whenever such a test has been made, this can be remembered until the information is not valid any more.

To this purpose we add additional edges to the abstraction that we label with $\sqsubseteq$ (and which we call *consistency edges*). We keep the invariant (that we will call *consistency invariant*) that whenever $a_1^* \rightarrow_{\sqsubseteq} a_2^*$, then $\mathit{Reach}(a_1^*, a_2) \sqsubseteq a_2^*$.

Moreover we use a procedure `propChange`(a) that, for every a' with $a \rightarrow a'$ deletes every edge $a \rightarrow_{\sqsubseteq} a'$. This allows us to change the while loop in the pruning algorithm as follows:

```
$\mathcal{A}^* \leftarrow$ copy of $\mathcal{A}$, all regions set to $\emptyset$, no initial labels, no edges
// from now on, for every abstract state $a$ of $\mathcal{A}$,
// we denote by $a^*$ the corresponding abstract state of $\mathcal{A}^*$
for all $a \in \mathcal{A}$, $a$ is initial
    $a^* \leftarrow \mathit{InitReach}(a)$
    if $a^* \neq \emptyset$ then
        mark $a^*$ as initial
        propChange($a^*$)
while there is a pair of abstract states $(a_1, a_2)$ in $\mathcal{A}$ with
    $a_1 \rightarrow a_2$, s.t. $a_1^* \not\rightarrow_{\sqsubseteq} a_2^*$ and $\mathit{Reach}(a_1^*, a_2) \not\subseteq a_2^*$ or
    ($a_1^* \not\rightarrow a_2^*$ and $\mathit{Reach}(a_1^*, a_2) \neq \emptyset$) do
        introduce an edge $a_1^* \rightarrow_{\sqsubseteq} a_2^*$
        if $a_1^* \not\rightarrow a_2^*$ in $\mathcal{A}^*$ then introduce an edge $a_1^* \rightarrow a_2^*$ into $\mathcal{A}^*$
        if $\mathit{Reach}(a_1^*, a_2) \not\subseteq a_2^*$ then $a_2^* \leftarrow a_2^* \uplus_{a_2} \mathit{Reach}(a_1^*, a_2)$
        propChange($a_2^*$)
return $\mathcal{A}^*$
```

Algorithms of such a type are known in the literature under them name "chaotic iteration" or "worklist algorithms". They have been used and studied mainly in the context of abstract interpretation [8,5,13] and constraint satisfaction [2,3].

Theorem 3. *Independent of the consistency edges of the input $\mathcal{A}$, the improved pruning algorithm computes the same result as the original one.*

Due to space restrictions we omit the proof of this theorem.

4.3 Incremental Refinement of Abstraction

Now observe that splitting, or dual pruning, only changes a part of the abstraction. Still, the pruning algorithms do a complete re-computation. This is not necessary, and in order to avoid it:

- We mark all abstract states for which we know, that a re-computation will not improve, with the mark `Cons` (the *consistency mark*).
- Whenever splitting or dual pruning changes an abstract state, we delete this consistency mark, and all consistency marks of states reachable from it.
- At the beginning of the pruning algorithm for all abstract states we reset the abstract state with the result of *InitReach* only if the consistency mark is not set. Abstract states with the consistency mark, retain the value from the input abstraction $\mathcal{A}$.

Since we do separate forward and backward pruning, we also need separate consistency marks for both cases. Splitting removes both consistency marks at the same time.

5 Implementation

The method introduced in this paper can be instantiated with various techniques for forming and representing abstract states, and computing reachability information. Nonetheless, in order to study the viability of the approach, we created a specific implementation that uses boxes (with floating point endpoints) for representing regions.

We initialize the abstraction by creating a single abstract state for each mode with box containing over-approximation of all reachable states in that mode. In our case, we have this box in the input. Abstract transitions are introduced for all pairs of abstract states contained in the *Jump* relation.

In the *Split* operation on boxes, we pick a splitting dimension of the box assigned to the region and we split the box into halves using this dimension. For picking the splitting dimension, a round-robin strategy has proved to be the useful heuristics [16].

5.1 Widening

Termination of the algorithm that uses boxes for region representation is ensured by doing all the computations on the *finite* set of floating point numbers. Hence there are only finitely many possibilities of changing boxes with $\uplus$, until a fixpoint is reached. This may in some cases lead to stuttering (i.e., many small improvements by close floating point numbers) and thus to a slow convergence to a fixpoint.

We designed a widening strategy to avoid stuttering and speed up the convergence of the algorithm. Here, widening is applied to the line $a_2^* \leftarrow (a_2^* \uplus_{a_2} \mathit{Reach}(a_1^*, a_2))$ of the algorithm, that we replace with $a_2^* \leftarrow \mathit{widening}(a_2, a_2^*, a_2^* \uplus_{a_2} \mathit{Reach}(a_1^*, a_2))$. Informally speaking, when a change from a_2^* to $a_2^* \uplus_{a_2} \mathit{Reach}(a_1^*, a_2)$ would be small compared to a_2, widening gives us region a bit bigger than $a_2^* \uplus_{a_2} \mathit{Reach}(a_1^*, a_2)$, but still smaller or equal than a_2.

Formally, the widening operates on each individual dimension separately (recall that in our implementation regions are represented by boxes, that is Cartesian products of intervals). Let b be the box $a_2^* \uplus_{a_2} \mathit{Reach}(a_1^*, a_2)$ and in the considered dimension i let $[\underline{a_2}_i, \overline{a_2}_i]$, $[\underline{a^*_2}_i, \overline{a^*_2}_i]$ and $[\underline{b}_i, \overline{b}_i]$ be the intervals of the boxes a_2, a^*_2 and b respectively. We will denote by $w_i := \overline{a_2}_i - \underline{a_2}_i$ the width of the box a_2 in its i-th dimension. For a given, pre-chosen constant $c \in [0, 1]$ (which we call *widening-constant*), the result of widening in the considered dimension is $[\underline{a_2}_i, \overline{a_2}_i] \cap [min(\underline{b}_i, \underline{a^*_2}_i - cw), max(\overline{b}_i, \overline{a^*_2}_i + cw)]$. Hence, widening depends on the width of the box in the original abstraction. After splitting the region, widening in the next execution of pruning algorithm adds smaller parts of the region. In our implementation, we use the widening constant $c = 1/16$.

Our previous approach for hybrid systems verification [16] computes abstractions based on an algorithm that can be viewed as is a special case of the method presented here that uses some extreme form of widening where $widening(a_2, a^*_2, a^*_2 \uplus_{a_2} Reach(a^*_1, a_2)) = Reach(a_1, a_2)$. In other words, the method does not accumulate reachability information at all, and immediately uses the weaker old abstract state a_1 instead of a^*_1. This results in a much simpler algorithm where instead of our while loop, reachability information is computed only once for each neighbor in the abstraction. However, as our computational experiments will show, different variants of the method presented in this paper show much better behavior than that particular instantiation, especially for hybrid systems with cyclic behavior.

5.2 Computation of Reachability Information

The way we compute reachability information in our implementation, as needed by functions *Reach* and *InitReach* is described in detail in older publications [16]. The main challenge is to come up with a way of handling the differentiation operator. One can over-approximate it to a purely polynomial constraint using Taylor expansion. Since *Reach* and *InitReach* only need reachability information in bounded regions, this results in bounds on all derivatives, and hence *Reach* and *InitReach* can be computed in a conservative way, even over unbounded time. Our current implementation only uses Taylor polynomials of degree one which corresponds to the mean-value theorem. This is a very crude over-approximation, which is sufficient for studying the behavior of the algorithm presented in this paper. In order to arrive at an efficient implementation, one would have to use Taylor polynomials of higher degrees, which is an easy, but tedious, implementation exercise.

The resulting constraint contains variables corresponding to derivatives and to the source points of trajectories and jumps. In order to arrive at a description of the set of reachable states, those have to be eliminated. In theory, one could use quantifier elimination procedures for this (cf. the notion of logical interpretation [18]). However, in the case of polynomials and real numbers, those do not scale in the problem dimension at all. Hence we simply project the boxes computed by interval constraint propagation [4] (we also have a generalization of this technique available [14]). We also have investigated an alternative method based on an over-approximation of Fourier-Motzkin elimination [10].

6 Computational Experiments

We studied the behavior of our implementation in two scenarios: First, the scenario where the abstraction refinement method of this paper is used for hybrid systems verification. And second, where it is used for computing abstractions of high-dimensional systems that can then be used for other purposes (e.g., running simulations for testing). Since we did not want to test the underlying reachability computation algorithms, but the incremental abstract computation algorithm introduced in this paper, we only used the highly over-approximating reachability computation described in Section 5.2. For practical analysis of concrete hybrid systems, one can use much more sophisticated reachability algorithms, adapted to the type of system at hand.

For studying the first scenario, we took benchmarks from our database (available on the web at `http://hsolver.sourceforge.net/benchmarks`). For each example, the behavior of a full verification cycle is described in Table 1. Here, the three main columns describe the three widening strategies described in Section 5.1, i.e., verification algorithm with (moderate) widening, without widening, and with the extreme widening strategy that corresponds to the previous algorithm from [16]. The column *Refine* represents the number of refinement steps of the overall algorithm for safety verification from Section 3. All timings were measured on PC with Intel Core 2 3.0GHz CPU and 4GB RAM. From the measured results we conclude that an extreme widening strategy does not pay off and can solve less benchmark examples than approaches with less aggressive widening. This is illustrated particularly nicely by the *cycle* benchmark, where verification finishes in one refinement step with the method presented in this paper, while the previous approach is not able to solve this benchmark at all. The reason is that for such benchmarks with cyclic system behavior, widening should not prevent the analysis of full system cycles.

We also conclude that the moderate widening strategy pays off since it causes only negligible run time increase, but removes stuttering in the *mutant* benchmark and does not increase the number of refinement steps in any benchmark.

For studying the second scenario we conducted computational experiments with high dimensional problems. For this purpose, we created two example problems using the parallel composition of several instances of simpler benchmarks. Our first high dimensional example contains 102 clock variables and one mode. The example was not verified in the time limit of one hour, however, the volume of all boxes in the abstraction was pruned from the size of approximately 8×10^{40} down to 1×10^{30}. Hence, our method resulted in an abstraction whose volume is 8×10^{10} times smaller, and that contains additional information about initial and unsafe states, and abstract transitions. The result can be used to guide testing or falsification [17], since we know that the pruned parts of the abstraction do not contain any error trajectory. Our second high dimensional benchmark is a benchmark with 100 variables with non-linear evolution, one clock variable and one mode. It was verified using thirteen refinement steps in 36 minutes. From the measured results in high dimension we conclude, that our technique for abstraction refinement is feasible also in case of high dimensional benchmarks.

Table 1. Experimental results

	F/B + widening		F/B reachability		orig. alg.	
Name	Refine	Time	Refine	Time	Refine	Time
1-flow	2	$< 1s$	2	$< 1s$	2	$< 1s$
2-tanks	2	$< 1s$	2	$< 1s$	9	$1s$
car	1	$< 1s$	1	$< 1s$	1	$< 1s$
circuit	101	$752s$	101	$706s$	N/A	
clock	15	$1s$	15	$1s$	16	$2s$
convoi-1	1	$< 1s$	2	$< 1s$	2	$< 1s$
convoi	19	$4s$	20	$4s$	N/A	
cycle	1	$< 1s$	1	$< 1s$	N/A	
eco	24	$18s$	24	$18s$	N/A	
fischer2	1	$< 1s$	1	$< 1s$	6	$40s$
focus	5	$< 1s$	5	$< 1s$	5	$< 1s$
hallstah	134	$463s$	134	$427s$	N/A	
mixing	1	$< 1s$	1	$< 1s$	1	$< 1s$
mutant	2	$< 1s$	N/A		N/A	
sinusoid	118	$121s$	118	$109s$	N/A	
van-der-pole	1	$< 1s$	1	$< 1s$	2	$< 1s$

The first benchmark we used was the parallel composition of 34, three-variable *1-flow* benchmarks from our database. The second benchmark was a parallel composition of 50 three-variable *clock* benchmarks, where all the instances share the common third clock variable, but have separate variables for non-linear evolution. The original clock benchmark needs fifteen refinement steps including fifteen splitting steps. Splitting in such a high dimension would create huge abstractions, so we have simplified the instances of the benchmark, changing the constant in the differential equation from $\dot{x_1} = -5.5x_2 + x_2^2$ to $\dot{x_1} = -5.5x_2 + 0.3x_2^2$. We believe, that instantiation of the technique with a reachability computation method that requires less splitting steps allows verification of the benchmark without this simplification.

7 Conclusion

In this paper we introduced a technique for computing abstractions of hybrid systems that can handle arbitrary hybrid systems for which certain reachability computation algorithms are provided. The abstractions are computed in such a way that they are as succinct as possible. Computational experiments confirm the usefulness of the approach. Especially, the approach can compute information for high-dimensional systems that can be used for guiding testing or falsification. In future work we will instantiate this technique with different methods for reachability computation and we will try to create an algorithm that provably terminates for all robust inputs [9,15].

References

1. Alur, R., Dang, T., Ivančić, F.: Predicate abstraction for reachability analysis of hybrid systems. Trans. on Embedded Computing Sys. 5(1), 152–199 (2006)
2. Apt, K.R.: The essence of constraint propagation. Theoretical Computer Science 221(1–2), 179–210 (1999)
3. Apt, K.R.: The role of commutativity in constraint propagation algorithms. ACM Transactions on Programming Languages and Systems 22(6), 1002–1036 (2000)
4. Benhamou, F., Granvilliers, L.: Continuous and interval constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming. ch.16, pp. 571–603. Elsevier, Amsterdam (2006)
5. Bourdoncle, F.: Efficient chaotic iteration strategies with widenings. In: Bjørner, D., Broy, M., Pottosin, I. (eds.) FMP&TA 1993. LNCS, vol. 735, pp. 128–141. Springer, Heidelberg (1993)
6. Brückner, I., Dräger, K., Finkbeiner, B., Wehrheim, H.: Slicing abstractions. Fundamenta Informaticae 89(4), 369–392 (2008)
7. Clarke, E., Fehnker, A., Han, Z., Krogh, B., Ouaknine, J., Stursberg, O., Theobald, M.: Abstraction and counterexample-guided refinement in model checking of hybrid systems. Int. J. of Foundations of Comp. Sc. 14(4), 583–604 (2003)
8. Cousot, P., Cousot, R.: Automatic synthesis of optimal invariant assertions: Mathematical foundations. In: Proceedings of the 1977 Symposium on Artificial Intelligence and Programming Languages, pp. 1–12 (1977)
9. Damm, W., Pinto, G., Ratschan, S.: Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. International Journal of Foundations of Computer Science (IJFCS) 18(1), 63–86 (2007)
10. Dzetkulič, T., Ratschan, S.: How to capture hybrid systems evolution into slices of parallel hyperplanes. In: ADHS 2009: 3rd IFAC Conference on Analysis and Design of Hybrid Systems, pp. 274–279 (2009)
11. Frehse, G., Krogh, B.H., Rutenbar, R.A.: Verifying analog oscillator circuits using forward/backward abstraction refinement. In: DATE 2006: Design, Automation and Test in Europe (2006)
12. Henzinger, T.A., Ho, P.-H.: A note on abstract interpretation strategies for hybrid automata. LNCS, vol. 999, pp. 252–264 (1995)
13. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer, Heidelberg (1999)
14. Ratschan, S.: Efficient solving of quantified inequality constraints over the real numbers. ACM Transactions on Computational Logic 7(4), 723–748 (2006)
15. Ratschan, S.: Safety verification of non-linear hybrid systems is quasi-semidecidable. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 397–408. Springer, Heidelberg (2010)
16. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation based abstraction refinement. ACM Transactions in Embedded Computing Systems 6(1) (2007)
17. Ratschan, S., Smaus, J.-G.: Finding errors of hybrid systems by optimising an abstraction-based quality estimate. In: Dubois, C. (ed.) TAP 2009. LNCS, vol. 5668, pp. 153–168. Springer, Heidelberg (2009)
18. Tiwari, A., Gulwani, S.: Logical interpretation: Static program analysis using theorem proving. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 147–166. Springer, Heidelberg (2007)

Composing Stability Proofs for Hybrid Systems

Corina Mitrohin and Andreas Podelski

University of Freiburg
Department of Computer Science
Freiburg, Germany
{mitrohin,podelski}@informatik.uni-freiburg.de

Abstract. A recent automatic proof method for the region stability of a hybrid system is based on the reachability analysis for a transformed hybrid system with double dimensionality (the transformation duplicates each of the continuous variables). We propose a new method which composes the reachability analyses for a sequence of transformed hybrid systems with essentially the same dimensionality (each transformation in the sequence duplicates one of the continuous variables). The new method thus trades the double dimensionality for the number of reachability analyses.

1 Introduction

Region stability is a correctness property for a hybrid system which expresses that each trajectory of the hybrid system eventually reaches a time point such that, from this time point on, the states of the trajectory remain in a specified region [4,5,6]. This property can be expressed by the finiteness of certain *snapshot sequences* of the hybrid system (certain sequences of states on a trajectory; the snapshot states lie outside the region although states between the snapshots lie possibly inside). The existing proof method for region stability computes a representation of the snapshot sequences of the hybrid system via the reachability analysis of a transformed hybrid system with double dimensionality [4,5,6]. This duplication of the dimensionality of the hybrid system is a severe bottleneck of the proof method.

We propose to split the computation of the representation of the snapshot sequences of the system. We sequentially compute *parts* of the snapshot sequences, that is sequences of snapshots of only one continuous variable. If we can prove that one of these partial snapshot sequences are finite (by proving that the binary reachability relation between the snapshots is well-founded, using an automatic tool like RankFinder [7]), we can conclude that the system is stable. If no, we continue by computing partial snapshot sequences for other variable, combine the results of the two analysis, and try again to prove the finiteness of the snapshots sequences, now made out of two variables. We continue this procedure until either the stability could be proven (as consequence of well-foundedness of a binary reachability relation between some partial snapshots), or we have computed the partial snapshot sequences for all continuous variables of the hybrid system.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 286–300, 2011.

The conjunction of all partial snapshot sequences describes the same snapshot sequences as delivered by the analysis of the system with double dimensionality.

In [2], the authors present a method to incorporate the result of a first analysis (of an abstraction of the original system), into a second analysis (of another abstraction of the same original system). Our present work differs in that it decomposes the reachability analysis inside the stability proof method itself, and obtains the same result as the monolithic analysis.

In summary, we present a new method to reduce the computational effort needed for the stability proof. We propose to compose the results of a sequence of analyses of hybrid systems obtained each by incrementing the number of variables by just 1. We thus trade the dimension explosion for the complexity of the composition of the stability proofs.

2 Motivation

In this section we will illustrate the idea of the present work on the basis of a simple example. In Figure 1 we have an hybrid system with two continuous variables, x and y, and two locations, ℓ_1 and ℓ_2. The system is stable wrt. region $x \leq 0$, that is, each trajectory eventually reaches the region $x \leq 0$ and remains there forever. If we want to prove stability following the method from [5], we will compute a representation of the snapshot sequences of the system, that is, a binary reachability relation between pair of states along the same trajectories, and try to prove well-foundedness for it. The well-foundedness of the binary reachability relation implies then finiteness of the snapshot sequences and thus stability [5]. The binary reachability relation is computed by reachability analysis of a transformed system which duplicates all continuous variables. In the case of the system from Figure 1, we observe that the behavior and the constraints on y do not influence the evolution of x, and thus are not relevant for the stability property. In such a case it is enough to compute the binary reachability relation between the x-snapshots (that is, the values of the variable x) along a trajectory. To compute the representation of the binary reachability relation we can use a reachability tool like PHAVer [3], and we obtain a symbolic representation of the reachable states, as are the constraints from (1). The variables d and flag are auxiliary variables needed for the generation of snapshot sequences. Variable d is an auxiliary clock variable which ensures that there is at least δ time distance between the snapshots (in our examples δ is set to $\frac{1}{100}$). The atomic propositions of the form $\mathsf{flag} == \ell_i^k,\ k \in \{1, 2, 3, 4\}$, describe the type of snapshot sequence being generated: $\mathsf{flag} == \ell_i^1$ will denote a sequence of snapshots on monotone ascending flows of the location ℓ_i, $\mathsf{flag} == \ell_i^2$ a sequence of snapshots on monotone descending flows of the location ℓ_i, $\mathsf{flag} == \ell_i^3$ a sequence of extremal snapshots in the location ℓ_i, and $\mathsf{flag} == \ell_i^4$ a sequence of entry points for the location ℓ_i, that is, points just after a discrete jump happened. In Section 5 we describe in more detail the differe

The x-snapshot sequences for the system depicted in Figure 1, computed while taking into consideration the evolution of variable y, are described by the

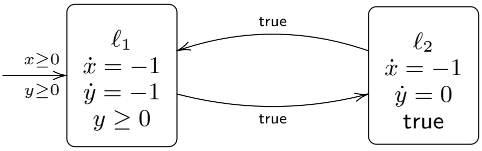


Fig. 1. Example of a hybrid system with two continuous variables, x and y. The system is stable with respect to $x \leq 0$. For the stability proof it is enough to show the well-foundedness of x-snapshot sequences, if their representation is computed taking into consideration the evolution of the variable y, too.

constraints from (1). We observe that each constraint of the disjunction describes a well-founded relation in $\text{`}x$ and $x$. The variable $\text{`}x$ denotes the *old value* of x.

$$\begin{array}{l}
\text{`}x \geq x + \mathsf{d} \ \wedge \ y \geq x \ \wedge \ 100\mathsf{d} > 1 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_2^4 \quad \vee \\
\text{`}x \geq x + \mathsf{d} \ \wedge \ 100\mathsf{d} > 1 \ \wedge \ y + \mathsf{d} \geq \text{`}x \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_2^4 \quad \vee \\
\text{`}x \geq x + \mathsf{d} \ \wedge \ y \geq x + d \ \wedge \ 100\mathsf{d} > 1 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_1^4 \quad \vee \\
x + \mathsf{d} == \text{`}x \ \wedge \ 100\text{`}x > 100x + 1 \ \wedge \ y \geq x \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_1^3 \quad \vee \\
y + \mathsf{d} == \text{`}x \ \wedge \ y \geq x \ \wedge \ 100\text{`}x > 100y + 1 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_2^4 \quad \vee \\
x + \mathsf{d} == \text{`}x \ \wedge \ 100\text{`}x > 100x + 1 \ \wedge \ y \geq \text{`}x \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_2^2 \quad \vee \\
x + \mathsf{d} == \text{`}x \ \wedge \ x == y \ \wedge \ 100\text{`}x > 100x + 1 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_1^3
\end{array} \qquad (1)$$

The next step is to consider systems like presented in Figure 2. In location ℓ_1, the evolution of x allows no useful reasoning for the stability proof. Indeed, if we compute the relation between the x-snapshots, we will obtain the constraints from (2). We observe that the second and the third disjunct describe a non well-founded relation.

$$\begin{array}{l}
\text{`}x \geq x + \mathsf{d} \ \wedge \ 100\mathsf{d} > 1 \ \wedge \ y \geq 0 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_1^4 \quad \vee \\
\text{`}x \geq x \ \wedge \ 100\mathsf{d} > 1 \ \wedge \ y \geq 0 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_2^4 \quad \vee \\
\text{`}x == x \ \wedge \ 100d > 1 \ \wedge \ y \geq 0 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_1^1 \quad \vee \\
x + d ==\text{`}\, x \ \wedge \ 100\text{`}x > 100x + 1 \ \wedge \ y \geq 0 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_2^3
\end{array} \qquad (2)$$

But, we can compute the binary reachability relation between the y-snapshot sequences, too. These are described by the constraints from (3). Again, not all constraints describe well-founded relations. But if we consider now the conjunction between the constraints from (2) and (3), conjunction which describes the binary reachability relation between the snapshots of the original system (this will be proved later in this paper), we obtain well-founded relations.

$$\begin{array}{l}
\text{`}y \geq y \ \wedge \ 100\mathsf{d} > 1 \ \wedge \ y \geq 0 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_1^4 \quad \vee \\
\text{`}y \geq y + d \ \wedge \ 100\mathsf{d} > 1 \ \wedge \ y \geq 0 \ \wedge \ y \geq 0 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_2^4 \quad \vee \\
y + d ==\text{`}\, y \ \wedge \ 100\text{`}y > 100y + 1 \ \wedge \ y \geq 0 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_1^1 \quad \vee \\
y ==\text{`}\, y \ \wedge \ 100\mathsf{d} > 1 \ \wedge \ y \geq 0 \ \wedge \ x > 0 \ \wedge \ \mathsf{flag} == \ell_2^3
\end{array} \qquad (3)$$

Of course, we can follow the approach from [5] and directly compute the binary reachability by reachability analysis on the transformed hybrid system with

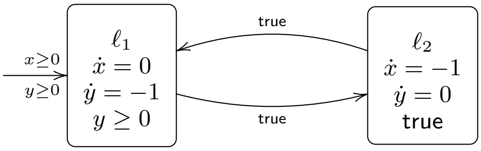


Fig. 2. Example of a hybrid system with two continuous variables, x and y. The system is stable with respect to $x \leq 0$. For the stability proof both x-snapshot sequences and y-snapshot sequences are needed.

double dimensionality. This analysis will return the constraints from (4). We observe that these constraints describe the same snapshot sequences as the constraints from (2) and (3). The drawback is that the reachability analysis is done on a hybrid system with $2n(+2n)$ variables. If we compute sequentially the x-snapshot sequences like previously described in this section, the reachability analysis is done on a system with $n+1(+2)$ continuous variables. In worst case (as for the system from Figure 2), we carry out n reachability analysis on systems of dimension $n+1(+2)$. In best case (as for the system from Figure 1), we do reachability analysis for *only one* system with $n+1(+2)$ continuous variables.

$$\begin{array}{l}
{}^{\backprime}x \geq x + d \;\wedge\; {}^{\backprime}y \geq y \;\wedge\; 100\mathsf{d} > 1 \;\wedge\; y \geq 0 \;\wedge\; x > 0 \;\wedge\; \mathsf{flag} == \ell_1^4 \vee \\
x + d == {}^{\backprime}x \wedge 100{}^{\backprime}x > 100x + 1 \wedge {}^{\backprime}y \geq y \wedge y \geq 0 \wedge x > 0 \wedge \mathsf{flag} == \ell_1^4 \vee \\
{}^{\backprime}x \geq x \;\wedge\; {}^{\backprime}y \geq y + d \;\wedge\; 100\mathsf{d} > 1 \;\wedge\; y \geq 0 \;\wedge\; x > 0 \;\wedge\; \mathsf{flag} == \ell_2^4 \vee \\
y + d == {}^{\backprime}y \wedge {}^{\backprime}x \geq x \wedge 100{}^{\backprime}y > 100y + 1 \wedge y \geq 0 \wedge x > 0 \wedge \mathsf{flag} == \ell_2^4 \vee \\
y + d == {}^{\backprime}y \wedge x == {}^{\backprime}x \wedge 100{}^{\backprime}y > 100y + 1 \wedge y \geq 0 \wedge x > 0 \wedge \mathsf{flag} == \ell_1^1 \vee \\
x + d == {}^{\backprime}x \wedge y == {}^{\backprime}y \wedge 100{}^{\backprime}x > 100x + 1 \wedge y >= 0 \wedge x > 0 \wedge \mathsf{flag} == \ell_2^3
\end{array} \tag{4}$$

3 Preliminaries

A *hybrid system* [1] is formally a tuple $\mathcal{H} = (Loc, V, Init, R^{\mathsf{cont}}, R^{\mathsf{disc}}, Inv)$ defining

- the finite set of locations Loc,
- the set of continuous variables $V = \{v_1, \ldots, v_n\}$,
- the initial condition, given by the constraint $Init(\ell)$ for each location ℓ,
- the continuous transition relation, given by the expression $e = R^{\mathsf{cont}}(\ell)(v)$ for each continuous variable v and each location ℓ; the expression e (in the variables $v_1, \ldots, v_n$) is used in the differential equation $\dot{v} = e$ that defines the flow of the continuous variable v in the location ℓ,
- the discrete transition relation, given by a set R^{disc} of transitions; a transition is formally a tuple (ℓ, g, ξ, ℓ') defining
  - the source location ℓ and the target location ℓ',
  - the guard, given by a constraint g,
  - the update, given by a (possibly empty) set ξ of assignments $v := e$ of expressions to continuous variables,

- the invariant, given by the constraint $Inv(\ell)$ for each location ℓ.

Examples of a hybrid systems are given in Section 2.

A *state* of the hybrid system $\mathcal{H}$ is a tuple $(\ell, x_1, \ldots, x_n)$ consisting of a location ℓ in *Loc* and values of the continuous variables in V. A set of states is also called a *region*. In what follows, we will consider only *interval regions*, i.e., regions φ described through $\varphi \equiv v \in [x_{min}, x_{max}]$, where v is a continuous variable of the hybrid system and x_{min} and x_{max} are constants (including $\pm\infty$) with $x_{min} < x_{max}$.

We now give the formal definition of the semantics of a hybrid system, in the form of the set of its runs. We write $\mathcal{T}$ for the set of all continuous time points which are denoted by non-negative real values, i.e., $\mathcal{T} = \mathrm{I\!R}_0^+$. A trajectory τ assigns to every time point t in $\mathcal{T}$ a location and a valuation of the variables v in V. Formally, a trajectory τ is a tuple

$$\tau = (\hat{\ell}, \hat{v}_1, \ldots, \hat{v}_n)$$

of functions $\hat{\ell} : \mathcal{T} \rightarrow Loc$ (for the current location) and $\hat{v} : \mathcal{T} \rightarrow \mathrm{I\!R}$ (for the current value of the variable v in V), such that there exists an infinite sequence of switching time points,

$$(t_i)_{i \in \omega} \in \mathcal{T}^{\omega}$$

which starts in 0 and is strictly increasing, i.e., $t_0 = 0$ and $t_i < t_{i+1}$, such that the following five conditions hold:

- "non-zenoness"
$$\forall t \in \mathcal{T}\ \exists i :\ t \leq t_i \tag{5}$$
- "switching time"
$$\forall i\ \forall t \in [t_i, t_{i+1}) : \hat{\ell}(t) = \hat{\ell}(\tau_i) \tag{6}$$
- "continuous evolution"
$$\forall v \in V\ \forall i\ \forall t \in [t_i, t_{i+1}) : \quad \frac{d}{dt}\hat{v}(t) = e[\hat{v}_1, \ldots, \hat{v}_n](t) \tag{7}$$
where $e = R^{\mathsf{cont}}(\ell)(v)$ with $\ell = \hat{\ell}(t_i)$,
- "invariants"
$$\forall i\ \forall t \in [t_i, t_{i+1}) :\ (\hat{v}_1(t), \ldots, \hat{v}_n(t)) \models Inv(\ell) \tag{8}$$
where $\ell = \hat{\ell}(t_i)$,
- "discrete transition firing"
$$\begin{array}{l} \forall i\ \exists\ (\ell, g, \xi, \ell') \in R^{\mathsf{disc}} : \\ \qquad \hat{\ell}\ (t_i) = \ell \\ \qquad \hat{\ell}\ (t_{i+1}) = \ell' \\ \qquad \exists\ \sigma : V \rightarrow \mathrm{I\!R}\ \forall v \in V : \\ \qquad\qquad \sigma(v) = \lim\limits_{u \rightarrow t_{i+1}} \hat{v}(u) \\ \qquad\qquad \sigma \models g \\ \qquad\qquad \hat{v}(t_{i+1}) = \begin{cases} \sigma(e), & \text{if } v := e \in \xi \\ \sigma(v), & \text{otherwise.} \end{cases} \end{array} \tag{9}$$

Condition (5) states that we do not allow Zeno behavior. The time sequence $(t_i)_{i\in\omega}$ identifies the time points where location switches may occur, which is expressed in Condition (6). Only at those points discrete transitions may be taken. Condition (7) expresses that the dynamics of the continuous variables obeys their respective differential equations. Condition (8) requires that for each location the valuation of continuous variables satisfies the local invariant while staying in that location. Condition (9) expresses that whenever a discrete transition is taken, variables may be assigned new values, obtained by evaluating the right-hand side of the respective assignment using the previous values of variables. If there is no such assignment, the variable maintains its previous value, which is determined by taking the limit of the trajectory of the variable as t converges to the switching time t_{i+1}.

A hybrid system is *stable* with respect to a region φ if for every trajectory τ there exists a point in time t_τ such that from then on, the trajectory is always in the region φ.

$$\forall\ \tau\ \exists t_\tau\ \forall t > t_\tau :\ \tau(t) \in \varphi$$

For a given hybrid system $\mathcal{H}$ and a region φ, a snapshot sequence is defined like follows.

A *snapshot sequence*

$$s_0,\ s_1,\ s_2, \ldots$$

is a sequence of states such that

(i) $\exists \tau\ \forall i\ \exists t_i :\ s_i = \tau(t_i)$,
(ii) $\forall i : s_i \notin \varphi$,
(iii) $\exists \delta > 0\ \forall i :\ t_{i+1} - t_i > \delta$.

A *snapshot* is thus a state of the hybrid system which does not lie in the region φ.

Definition 1 (v_i-Snapshot Sequence). *The projection of a snapshot on the first component (stays for the discrete location) and the i-th variable v_i, is called a v_i-*snapshot*.*

If we consider the projection on the first component and the i-th variable of each state of a snapshot sequence, we obtain a sequence

$$(\ell_0, x_i^0), (\ell_1, x_i^1), (\ell_2, x_i^2), \ldots$$

*called a v_i-*snapshot sequence*.*

In the remainder of this paper, we will distinguish between three different types of snapshot sequences [5]. A *snapshot sequence on monotonic flow* is the discretization of a monotonic trajectory without jumps. *Sequences of extremal points* are sequences of points at which a trajectory changes its monotonicity behavior during a continuous flow. *Sequences of entry-points* are made of points just after a discrete jump happened. These three kinds of snapshot sequences can be naturally extended to the notion of v_i-snapshot sequences by considering the projection of the snapshot sequence on the variable v_i.

4 Complexity Reduction for Stability Proof

First, we recall the sound and complete stability proof method presented in [5].

Theorem 1 ([5]). *A hybrid system $\mathcal{H}$ is stable with respect to the region φ if and only if the following three conditions hold altogether.*

(1) There is no infinite snapshot sequence such that
  (i) no entry points lies between two states of the sequence and
  (ii) no extremal-point lies between two states of the sequence.
(2) There is no infinite snapshot sequence such that all states of the sequence are extremal points.
(3) There is no infinite snapshot sequence such that all states of the sequence are entry-points.

Once this sound and complete result was established, the challenge is to first compute a representation of the sequences of snapshot sequences, and second, prove their finiteness. In [5], the computation of an effective representation of the set of the snapshot sequences is done via reachability analysis of a transformation $\mathcal{T}(\mathcal{H})$ of the hybrid system $\mathcal{H}$. Since this transformation duplicates all continuous variables of $\mathcal{H}$, the analysis of $\mathcal{T}(\mathcal{H})$, and thus the computation of the set of snapshot sequences, becomes difficult even for low dimensions of $\mathcal{H}$.

It is trivial to observe that the finiteness of snapshot sequences is equivalent to the finiteness of v-snapshot sequences, since this property remains unchanged under the projection.

Remark 1. A hybrid system $\mathcal{H}$ is stable with respect to the region φ if and only if the following three conditions hold altogether.

(1) There is no $v \in V$ and there is no infinite v-snapshot sequence, such that
  (i) no entry points lies between two states of the sequence and
  (ii) no extremal-point lies between two states of the sequence.
(2) There is no $v \in V$ and there is no infinite v-snapshot sequence such that all states of the sequence are extremal points.
(3) There is no $v \in V$ and there is no infinite v-snapshot sequence such that all states of the sequence are entry-points.

In this paper we exploit the equivalence between the finiteness of snapshot sequences and the finiteness of v-snapshot sequences. We show how to iteratively compute v-snapshot sequences, duplicating only one continuous variable at a time. We define a simplified system transformation, $\mathcal{T}_v(\mathcal{H})$, for each $v \in V$. We then combine the different v-snapshot sequences such that, in worst case, we reobtain the set of snapshot sequences of $\mathcal{H}$. This procedure reduces the analysis of a $2n(+2)$-dimensional hybrid system to, in worst case, n analyses of a $n+1(+2)$-dimensional hybrid systems.

5 Compute Effective Representation of Snapshot Sequences

In [5,8], it is shown that it suffices to prove the finiteness of three types of snapshot sequences, since this is equivalent to the finiteness of the whole set of snapshot sequences. These three types of snapshot sequences are mentioned at the end of Section 3.

For the hybrid systems $\mathcal{H}$ and the region $\varphi \equiv v \in [v_{\min}, v_{\max}]$, let $v_i \in V$ be a variable of $\mathcal{H}$. To compute a representation of the v_i-snapshot sequences we use a simplified version of the system transformation from [5,8]. We denote the transformed system from [5,8] by $\mathcal{T}(\mathcal{H})$, and the system used to compute the v_i-snapshot sequences by $\mathcal{T}_{v_i}(\mathcal{H})$, for $v_i \in V$. The system $\mathcal{T}_{v_i}(\mathcal{H})$ will be defined later in this section.

Now, we recall the generation of the three types of snapshot sequences from [5,8], and point out the differences we made to compute the v_i-snapshot sequences.

Both systems $\mathcal{T}(\mathcal{H})$ and $\mathcal{T}_x(\mathcal{H})$ contain two auxiliary variables d and flag. The variable d is a clock variables and assures that at least δ units of time elapsed between two snapshots. The variable flag acts like a program counter and gives information about the type of snapshot sequence being generated.

Each state of the system $\mathcal{T}(\mathcal{H})$ corresponds to a pair of states $({}^{\backprime}s, s)$ of $\mathcal{H}$, with s a tuple of the form $s = (\ell, x_1, \ldots, x_n)$. Whenever the state s is reachable from the state ${}^{\backprime}s$ in the original system, $({}^{\backprime}s, s)$ is reachable in the transformed system. This property is referred as *binary reachability* to, i.e., a pair of states $({}^{\backprime}s, s)$ is called binary reachable in a hybrid system $\mathcal{H}$, if there exists a trajectory τ of $\mathcal{H}$ such that

- ${}^{\backprime}s$ is a state on τ at time point ${}^{\backprime}t$: ${}^{\backprime}s = \tau({}^{\backprime}t)$;
- s is a state on τ at time point t: $s = \tau(t)$;
- ${}^{\backprime}t < t$.

This property of $\mathcal{T}(\mathcal{H})$ is achieved by sampling the trajectories of $\mathcal{H}$. Technically, this is realized by duplicating all continuous variables of $\mathcal{H}$; we speak now about *old variables* ${}^{\backprime}v$, and *variables* v. Initially, the variables ${}^{\backprime}v$ and v behave similarly. After the elapse of a discretization step δ and if the actual values of v do not satisfy the region property, the old values ${}^{\backprime}v$ are frozen, and only the variables v continue evolving. After another passing of at least δ time units, and if again the variables v do not lie in the region, the variables v are frozen. The pair of states described by the valuations of the old variables ${}^{\backprime}v$ and the variables v, $(({}^{\backprime}\ell, {}^{\backprime}x_1, \ldots, {}^{\backprime}x_n), (\ell, x_1, \ldots, x_n))$, are binary reachable in $\mathcal{H}$.

The states of $\mathcal{T}_{v_i}(\mathcal{H})$ will be of the form $(\ell, {}^{\backprime}x_i, x_1, \ldots, x_n)$ with the property that whenever $(({}^{\backprime}\ell, {}^{\backprime}x_1, \ldots, {}^{\backprime}x_n), (\ell, x_1, \ldots, x_n))$ is binary reachable in $\mathcal{H}$, $(\ell, {}^{\backprime}x_i, x_1, \ldots, x_n)$ is reachable in $\mathcal{T}_{v_i}(\mathcal{H})$.

The v_i-snapshot pair $({}^{\backprime}x_i, x_i)$ is binary reachable in $\mathcal{H}$, if there exists a pair of states $({}^{\backprime}s, s)$ which is binary reachable in $\mathcal{H}$ and its projection on the i-th component is $({}^{\backprime}x_i, x_i)$.

Technically, we duplicate *only* the variable v_i; we have now the new variable $`v_i$. Initially, the variables $`v_i$ and v_i behave similarly. After the elapse of a discretization step δ and if the actual values of the variables v do not satisfy the region property, the old variable $`v_i$ is frozen, and only the unprimed variables continue evolving. After another passing of at least δ time units, and if again the variables v do not lie in the region, the unprimed variables are frozen, too.

Since we are interested only in three types of snapshot sequences, we informally describe how can we compute only these kinds of snapshots.

(v_i-)Sequences of extremal points: For the generation of sequences of extremal points for a location, we must assure that each snapshot corresponds to an extremal point of the trajectory of v (v is the variable occurring in the region description). This can be done by doing the discretization inside a location only when the derivative of v is 0.

(v_i-)Snapshots on monotone flows: For the generation of snapshots on monotone flows within a location, we must assure that each snapshot lies on an descending or ascending part of a trajectory, and that no direction change takes place in between. This can be done by testing the sign of the derivative of v, and discretizing as long as the sign remains uchanged.

(v_i)-Sequences of entry points: For the generation of sequences of entry points for a location, we must assure that each snapshot, that is, each discretization of the trajectory, corresponds to a state just after a discrete jump.

We now give the formal definition of the system $\mathcal{T}_x(\mathcal{H})$, in the style of the definition of $\mathcal{T}(\mathcal{H})$ [4].

Given a hybrid system

$$\mathcal{H} = (Loc, V, Init, R^{\mathsf{cont}}, R^{\mathsf{disc}}, Inv)$$

for which we want to prove stability with respect to the interval region

$$\varphi \quad \equiv \quad v \in [v_{min}, v_{max}] \;,$$

the transformed system $\mathcal{T}_{v_i}(\mathcal{H}),\ v_i \in V$ is given by

Locations: Each location ℓ_s of the original system corresponds to five locations $\ell_s^0, \ldots, \ell_s^4$ in the transformed system. We refer to the set of all locations from ℓ_1^k to ℓ_m^k as Loc^k,

$$Loc^k = \{\ell_1^k, \ldots, \ell_m^k\} \;,\; k \in \{0, 1, 2, 3, 4\} \,.$$

In addition, the transformed system has two locations ℓ_{init} and ℓ_{end}. Altogether, the set $Loc^{T_{v_i}}$ of locations of the transformed system consists of the following components:

$$Loc^{T_{v_i}} \;=\; \{\ell_{\mathsf{init}}\} \,\cup\, Loc^0 \,\cup\, Loc^1 \,\cup\, Loc^2 \,\cup\, Loc^3 \,\cup\, Loc^4 \,\cup\, \{\ell_{\mathsf{end}}\}$$

Variables: The set $V^{T_{x_i}}$ of variables of the transformed system contains all variables of V, the old v_i, $`v_i$, an additional variable flag, and the clock variable d.

$$V^{T_{v_i}} \;=\; \{`v_i\} \,\cup\, V \,\cup\, \{\mathsf{flag}, \mathsf{d}\}$$

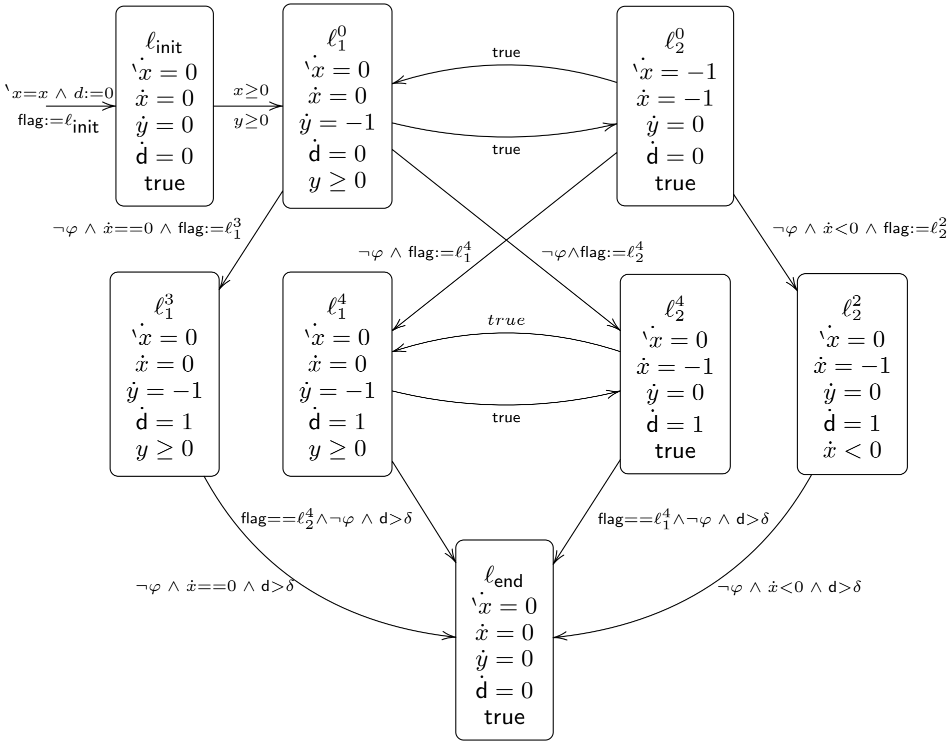


Fig. 3. System transformation $\mathcal{T}_x(\mathcal{H})$ for the computation of x-snapshot sequences for the hybrid system depicted in Figure 2. The stable region is $\varphi \equiv x \leq 0$.

Initial conditions: Initially, the variable $\grave{}v_i$ has the same value as v_i, the values of d and flag are set to 0 and ℓ_{init}, respectively; the system starts in the location ℓ_{init}.

$$\begin{aligned} Init^{T_{v_i}}(\ell_{\mathsf{init}}) \quad &\equiv \quad \grave{}v_i = v_i \ \wedge\ \mathsf{d} = 0 \ \wedge\ \mathsf{flag} = \ell_{\mathsf{init}} \\ Init^{T_{v_i}}(\ell) \quad &\equiv \quad \text{false} \quad\ \forall\, \ell \neq \ell_{\mathsf{init}} \end{aligned}$$

Continuous Transition Relation: First, in the locations ℓ_{init} and ℓ_{end} the flow of all continuous variables is 0.

$$\begin{aligned} R^{\mathsf{cont}}_{T_{v_i}}(\ell_{\mathsf{init}})(v) \quad &\equiv \quad \dot{v} = 0,\ v \in V^{T_{v_i}} \\ R^{\mathsf{cont}}_{T_{v_i}}(\ell_{\mathsf{end}})(v) \quad &\equiv \quad \dot{v} = 0,\ v \in V^{T_{v_i}} \end{aligned}$$

In each location ℓ_s^0 of Loc^0, the flow of the variables $v_1, \ldots, v_n$ in the transformed system is the same as the flow of $v_1, \ldots, v_n$ in the original system; the variable $\grave{}v_i$ behaves exactly like its unprimed version, that is the flow of $\grave{}v_i$ is equal to the flow of the original system after replacing the variable v_i by its pre-primed version $\grave{}v_i$.

$$
\begin{array}{rcl}
R^{\mathsf{cont}}_{T_{v_i}}(\ell^0_s)(v_j) & \equiv & R^{\mathsf{cont}}_{T_{v_i}}(\ell^0_s)(v_j)\ ,\ v_j \in V \\
R^{\mathsf{cont}}_{T_{v_i}}(\ell^0_s)({}^\backprime v_i) & \equiv & R^{\mathsf{cont}}_{T_{v_i}}(\ell^0_s)(v_i)[{}^\backprime v_i/v_i]
\end{array}
$$

In each location of $Loc^1 \cup \ldots \cup Loc^4$ the value of the variable ${}^\backprime v_i$ is fixed, i.e. its flow is constantly 0. The variables $v_1, \ldots, v_n$ continue evolving as before. The flow of the the clock variable d is 1 and the variable flag remains constant.

$$
\begin{array}{rcl}
R^{\mathsf{cont}}_{T_{v_i}}(\ell^k_s)({}^\backprime x_i) & \equiv & \dot{{}^\backprime v_i} = 0 \\
R^{\mathsf{cont}}_{T_{v_i}}(\ell^k_s)(x_j) & \equiv & R^{\mathsf{cont}}(\ell_s)(v_j)\ ,\ v_j \in V \\
R^{\mathsf{cont}}_{T_{v_i}}(\ell^k_s)(\mathsf{d}) & \equiv & \dot{\mathsf{d}} = 1 \\
R^{\mathsf{cont}}_{T_{v_i}}(\ell^k_s)(\mathsf{flag}) & \equiv & \dot{\mathsf{flag}} = 0
\end{array}
$$

Discrete Transition Firing: A discrete transition is possible from the location ℓ_{init} to any location ℓ^0_s of Loc^0 if the initial condition of the location ℓ_s of the original system $\mathcal{H}$ is fulfilled for the variables $(v_1, \ldots, v_n)$.

$$(\ell_{\mathsf{init}},\ Init(\ell_s),\ \varnothing,\ \ell^0_s)\ \in\ R^{\mathsf{disc}}_{T_{v_i}}$$

The second kind of discrete transitions are transitions between two locations of Loc^0 and between two locations of Loc^4, respectively. The condition for a jump between the location ℓ^0_s and the location ℓ^0_t for the variables $({}^\backprime v_i, v_1, \ldots, v_n)$ corresponds to the jump condition between the locations ℓ_s and ℓ_t of the original system $\mathcal{H}$ for the variables $(v_1, \ldots, v_n)$. Similarly a jump condition between the locations ℓ^4_s and ℓ^4_t of the transformed system corresponds to the jump condition from location ℓ_s to ℓ_t.

$$
\begin{array}{rcl}
(\ell^0_s, g, \xi \cup \xi[{}^\backprime v_i/v_i], \ell^0_t)\ \in\ R^{\mathsf{disc}}_{T_{v_i}} & \text{iff} & (\ell_s, g, \xi, \ell_t)\ \in\ R^{\mathsf{disc}} \\
(\ell^4_s, g, \xi, \ell^4_t)\ \in\ R^{\mathsf{disc}}_{T_{v_i}} & \text{iff} & (\ell_s, g, \xi, \ell_t)\ \in\ R^{\mathsf{disc}}
\end{array}
$$

To compute sequences on monotonic flows we allow jumps from $\ell^0_s \in Loc^0$ to $\ell^1_s \in Loc^1$ (and to $\ell^2_s \in Loc^2$, respectively) if the first derivative of v is greater than 0 (and less than 0, respectively) and if v does not lie in φ. During the jump the value of flag is set to the label ℓ^1_s (and ℓ^2_s, respectively).

$$
\begin{array}{c}
(\ell^0_s, \dot{v} > 0 \wedge \neg\varphi, \{\mathsf{flag} := \ell^1_s\}, \ell^1_s)\ \in\ R^{\mathsf{disc}}_{T_{v_i}} \\
(\ell^0_s, \dot{v} < 0 \wedge \neg\varphi, \{\mathsf{flag} := \ell^2_s\}, \ell^2_s)\ \in\ R^{\mathsf{disc}}_{T_{v_i}}
\end{array}
$$

Similarly, the transformed system can take a jump from a location $\ell^0_s \in Loc^0$ to a location $\ell^3_s \in Loc^3$ if the first derivative of v is 0 and if v does not lie in φ. During the jump the value of flag is set to the label ℓ^3_s.

$$(\ell^0_s, \dot{v} = 0 \wedge \neg\varphi, \{\mathsf{flag} := \ell^3_s\}, \ell^3_s)\ \in\ R^{\mathsf{disc}}_{T_{v_i}}$$

A jump from ℓ_s^1 to ℓ_{end} (and from ℓ_s^2 to ℓ_{end}, respectively) is possible if the first derivative of v is greater than 0 (and less than 0, respectively), if v does not lie in φ, and if if the system has at least spend time δ in the location ℓ_s^1 (and in ℓ_s^2, respectively).

$$\begin{aligned}(\ell_s^1, \dot{v} > 0 \wedge \neg\varphi \wedge \mathsf{d} > \delta, \varnothing, \ell_{\mathsf{end}}) &\in R_{T_{v_i}}^{\mathsf{disc}}\\ (\ell_s^2, \dot{v} < 0 \wedge \neg\varphi \wedge \mathsf{d} > \delta, \varnothing, \ell_{\mathsf{end}}) &\in R_{T_{v_i}}^{\mathsf{disc}}\end{aligned}$$

Similarly, the transformed system can jump from ℓ_s^3 to ℓ_{end} if the system has at least spend time δ in the location ℓ_s^3, if the first derivative of v is 0, and if v is not in φ.

$$(\ell_s^3,\ \dot{v} = 0\ \wedge\ \neg\varphi\ \wedge\ \mathsf{d} > \delta,\ \varnothing,\ \ell_{\mathsf{end}}) \in R_{T_{v_i}}^{\mathsf{disc}}$$

For the computation of pairs of entry-points of the original system we need a jump from a location $\ell_s^0 \in \mathit{Loc}^0$ to a location $\ell_t^4 \in \mathit{Loc}^4$ whenever the jump condition between the locations ℓ_s and ℓ_t of the original system $\mathcal{H}$ is possible but only if v is not in the region φ. During the jump the value of flag is set to the label ℓ_t^4 of the target location.

$$(\ell_s^0,\ g\ \wedge\ \neg\varphi,\ \xi \cup \{\mathsf{flag} := \ell_t^4\},\ \ell_t^4) \in R_{T_{v_i}}^{\mathsf{disc}} \quad \text{iff} \quad (\ell_s,\ g,\ \xi,\ \ell_t) \in R^{\mathsf{disc}}$$

The transformed system can jump from $\ell_s^4 \in \mathit{Loc}^4$ to ℓ_{end} if v is not in φ, the value of flag is t, and the jump condition from ℓ_s to ℓ_t of the original system $\mathcal{H}$ holds. Additionally we must guarantee the discretization width δ (for $\delta > 0$ constant). The condition $\mathsf{flag} = \ell_t^4$ ensures that the denoted binary relation represents sequences of entry-points such that all entry-points have the same location ℓ_t.

$$(\ell_s^4,\ g \wedge \mathsf{flag} = \ell_t^4 \wedge\neg\varphi \wedge \mathsf{d} > \delta,\ \varnothing,\ \ell_{\mathsf{end}}) \in R_{T_{v_i}}^{\mathsf{disc}} \quad \text{iff} \quad (\ell_s,\ g,\ \xi,\ \ell_t) \in R^{\mathsf{disc}}$$

Invariants: For the locations ℓ_{init} and ℓ_{end} the invariant condition is true.

$$\begin{aligned}\mathit{Inv}^{T_{v_i}}(\ell_{\mathsf{init}}) &\equiv \text{true}\\ \mathit{Inv}^{T_{v_i}}(\ell_{\mathsf{end}}) &\equiv \text{true}\end{aligned}$$

For a location ℓ_s^0 in Loc^0 the invariant condition over $`x_i, x_1, \ldots, x_n, \mathsf{flag}, \mathsf{d}$ is the same as the invariant condition of the original system $\mathcal{H}$ for ℓ_s over $x_1, \ldots, x_n$.

$$\mathit{Inv}^{T_{x_i}}(\ell_s^0) \equiv \mathit{Inv}(\ell_s)$$

For a location $\ell_s^1 \in \mathit{Loc}^1$ (and $\ell_s^2 \in \mathit{Loc}^2$, respectively) the invariant condition over $`v_i, v_1, \ldots, v_n, \mathsf{flag}, \mathsf{d}$ corresponds to the invariant condition of the original system $\mathcal{H}$ for ℓ_s over $v_1, \ldots, v_n$ in addition to the condition $\dot{v} > 0$ (and $\dot{v} < 0$, respectively).

$$\begin{aligned}\mathit{Inv}^{T_{v_i}}(\ell_s^1) &\equiv \mathit{Inv}(\ell_s)\ \wedge\ \dot{v} > 0\\ \mathit{Inv}^{T_{v_i}}(\ell_s^2) &\equiv \mathit{Inv}(\ell_s)\ \wedge\ \dot{v} < 0\end{aligned}$$

For a location $\ell_s^3 \in Loc^3$ or $\ell_s^4 \in Loc^4$ the invariant condition over $`v_i, v_1, \ldots, v_n, \mathsf{flag}, \mathsf{d}$ is the same as the invariant condition of the original system $\mathcal{H}$ for ℓ_s over $v_1, \ldots, v_n$.

$$\begin{array}{rcl} Inv^{T_{v_i}}(\ell_s^3) & \equiv & Inv(\ell_s) \\ Inv^{T_{v_i}}(\ell_s^4) & \equiv & Inv(\ell_s) \end{array}$$

We say that a state $s = (\ell, \ldots)$ is *reachable at* ℓ_{end} in $\mathcal{T}(\mathcal{H})$ (in $\mathcal{T}_v(\mathcal{H})$, respectively), if $\ell = \ell_{\mathsf{end}}$.

Figure 3 depicts the system transformation $\mathcal{T}_x(\mathcal{H})$ for the hybrid system from Figure 2.

Lemma 1. *The transformed systems $\mathcal{T}(\mathcal{H})$ and $\mathcal{T}_{v_i}(\mathcal{H})$ generate the same sequences of v_i-snapshots. Moreover, we have that*

$$\begin{array}{rl} \mathrm{Reach}(\mathcal{T}(\mathcal{H})) \downarrow_{\ell_{\mathsf{end}}} = & \{(\ell_{\mathsf{end}}, `x_1, \ldots, `x_n, x_1, \ldots, x_n, \mathsf{d}, \ell^k),\ k \in \{1,2,3,4\},\ \mathsf{d} > \delta \mid \\ & (\ell_{\mathsf{end}}, `x_i, x_1, \ldots, x_n, \mathsf{d}, \ell^k) \in \mathrm{Reach}(\mathcal{T}_{v_i}(\mathcal{H}))\} \end{array}$$

Proof. The proof of the lemma follows from the construction of $\mathcal{T}_{v_i}(\mathcal{H})$. By definition, the evolution of the variables $`v_i,\ v_i$ is the same in both systems $\mathcal{T}(\mathcal{H})$ and $\mathcal{T}_{v_i}(\mathcal{H})$. The guards of the discrete transition relation and the invariants are all conditions in the unprimed variables v, so they do not change the switching behavior of $\mathcal{T}_{v_i}(\mathcal{H})$. □

The reachable set at ℓ_{end} in $\mathcal{T}_{v_i}(\mathcal{H})$ describes the binary reachability relation between the v_i-snapshots of the original system $\mathcal{H}$, result stated in the lemma bellow.

Lemma 2. *A pair $((\ell, `x_i), (\ell, x_i))$ of $v_i$-snapshots is binary reachable in $\mathcal{H}$ iff there exists a state $\tilde{s}$ reachable at $\ell_{\mathsf{end}}$ in $\mathcal{T}_{v_i}(\mathcal{H})$ such that $\tilde{s} = (\ell_{\mathsf{end}}, `x_i, \ldots, x_i, \ldots, \ell^k)$, with $k \in \{1,2,3,4\}$.*

Proof. The proof of the lemma follows from Lemma 1 and the results established in [5,8].

The pair $((\ell, `x_i), (\ell, x_i))$ is binary reachable in $\mathcal{H}$ iff there exists a pair of states of $\mathcal{H}$, $(`s, s) = ((\ell, `x_1, \ldots, `x_n), (\ell, x_1, \ldots, x_n))$, such that s is reachable from $`s$ in $\mathcal{H}$, and $`s$ and s does not lie in the region φ.

From [4,5,8], we know that the reachability relation of $\mathcal{T}(\mathcal{H})$ is the binary reachability relation between the snapshots of $\mathcal{H}$. This means that the state $(\ell_{\mathsf{end}}, `x_1, \ldots, `x_n, x_1, \ldots x_n, \mathsf{d}, \ell^k)$, $k \in \{1,2,3,4\}$, $\mathsf{d} > \delta$, is reachable at ℓ_{end} in $\mathcal{T}(\mathcal{H})$. From Lemma 1, we have that $\tilde{s} = (\ell_{\mathsf{end}}, `x_i, x_1, \ldots x_n, \mathsf{d}, \ell^k)$ is reachable at ℓ_{end} in $\mathcal{T}_{v_i}(\mathcal{H})$, too. □

In [4], a soundness result is established between the well-foundedness of the reachability relation of $\mathcal{T}(\mathcal{H})$ and the region stability property of $\mathcal{H}$. More exactly, the well-foundedness of the binary reachability relation of $\mathcal{H}$ (that is, the reachability relation of $\mathcal{T}(\mathcal{H})$) implies region stability. From Lemma 2, we have the same soundness result for the conjunction of the reachability relations of the systems $\mathcal{T}_{x_i}(\mathcal{H})$. Moreover, we observe that the well-foundedness of a single relation of $\text{Reach}(\mathcal{T}_{v_i}(\mathcal{H}))$ is a stronger condition for region stability, since it implies the well-foundedness of $\text{Reach}(\mathcal{T}(\mathcal{H}))$ and so the region stability of $\mathcal{H}$. This observation is summarized by the following lemma.

Lemma 3. *If there exists an $v_i \in V$ such that the relations describing the v_i-snapshot sequences are well-founded, then the hybrid system $\mathcal{H}$ is stable wrt. φ.*

Algorithm 1. Input: Hybrid system $\mathcal{H}$ and φ **Output:** '$\mathcal{H}$ is stable wrt to φ' or 'stability unknown'

```
S ← ∅
for ℓ ∈ Loc do
    for k = 1 to 4 do
        $c_l^k \leftarrow$ true
    end for
end for
repeat
    chose $v_i$ from $V \setminus S$
    $S \leftarrow S \cup \{v_i\}$
    Build $\mathcal{T}_{v_i}(\mathcal{H})$; $C_{\mathsf{aux}} \leftarrow \text{Reach}(\mathcal{T}_{v_i}(\mathcal{H}))$
    if $C_{\mathsf{aux}}$ is disjunctive well-founded then
        return $\mathcal{H}$ is stable wrt. $\varphi$
    end if
    for each $c_l^k$, $c_l^k$ not disjunctive well-founded, do
        $c_l^k \leftarrow c_\ell^k \wedge C_{\mathsf{aux}}$
    end for
until $V \setminus S = \varnothing$ or exists a not well-founded $c_l^k$
if for all $\ell \in Loc$, $k = \overline{1,4}$, $c_l^k$ is disjunctive well-founded then
    return $\mathcal{H}$ is stable wrt. $\varphi$
else
    return stability unknown
end if
```

Our approach to prove stability of $\mathcal{H}$ wrt. to a region $x \in [x_{min}, x_{max}]$ is summarized in Algorithm 1. We denote by c_ℓ^k, $\ell \in Loc$, $k \in \{1,2,3,4\}$, the constraints describing the different types of snapshot sequences for the location ℓ. At the beginning all these constraints are true. Our goal is to prove by iterative computation of v_i-snapshot sequences that each such constraint describes a disjunctive well-founded relation. Let $C = \bigvee_{\ell \in Loc} \bigvee_{k=\overline{1,4}} c_\ell^k$.

We first start by choosing a variable v_i from V and construct the transformed hybrid system $\mathcal{T}_{v_i}(\mathcal{H})$. A reachability tool like PHAVer [3] delivers a symbolic representation of $\text{Reach}(\mathcal{T}_{v_i}(\mathcal{H}))$ of the reachability set in form of a disjunction of conjunctions of constraints over the variables of $\mathcal{T}_{v_i}(\mathcal{H})$. If $\text{Reach}(\mathcal{T}_{v_i}(\mathcal{H}))$ is disjunctive well-founded (property which can be checked using automated tools like [7]), then the system is stable by Lemma 3 (such a case is the hybrid system from Figure 1; the x-snapshot sequences suffices to prove stability wrt. $x \leq 0$). If this is not the case, let $C := C \wedge \text{Reach}(\mathcal{T}_{v_i}(\mathcal{H}))$. Now, if C is a disjunctive well-founded (in $(({}^\backprime v_1, \ldots, {}^\backprime v_i), (v_1, \ldots, v_i))$), then we can stop with the answer 'systems is stable'. If no, we continue by choosing a new variable v_{i+1} from V, and repeat the procedure. For the systems from Figure 2, we need to compute both $\text{Reach}(\mathcal{T}_x(\mathcal{H}))$ and $\text{Reach}(\mathcal{T}_y(\mathcal{H}))$ and to build the conjunction between the two sets of partial snapshots.

The algorithm stops if either the system is stable (that is, C is a disjunction of well-founded relations), or there there is no new variable in V to be chosen. In the last case, the answer will be 'stability unknown'.

6 Conclusion

In this paper, we have presented a new method to compute and combine parts of snapshot sequences of a hybrid system. We have obtained an approach where one can incorporate the results of at most n reachability analyses, and thus avoid the analysis of a hybrid system with double dimensionality.

Our approach is applicable to all classes of hybrid systems. The computation of the representation of snapshot sequences is restricted to class of systems which can be handled by the reachability tool (like for example PHAVer [3]).

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
2. Bogomolov, S., Mitrohin, C., Podelski, A.: Composing reachability analyses of hybrid systems for safety and stability. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 67–81. Springer, Heidelberg (2010)
3. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past hyTech. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 258–273. Springer, Heidelberg (2005)
4. Podelski, A., Wagner, S.: Model checking of hybrid systems: From reachability towards stability. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 507–521. Springer, Heidelberg (2006)
5. Podelski, A., Wagner, S.: Region stability proofs for hybrid systems. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 320–335. Springer, Heidelberg (2007)
6. Podelski, A., Wagner, S.: A sound and complete proof rule for region stability of hybrid systems. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 750–753. Springer, Heidelberg (2007)
7. Rybalchenko, A.: RankFinder, `http://www.mpi-inf.mpg.de/rybal/rankfinder`
8. Wagner, S.: Stability Proofs for Hybrid Systems. PhD thesis, Universität des Saarlandes (2007)

Rigorous Discretization of Hybrid Systems Using Process Calculi

Sven Schneider and Uwe Nestmann

Technische Universität Berlin

Abstract. In the context of a hybrid process calculus, we present a formal discretization procedure that abstracts a class of hybrid systems to simply timed systems while preserving observational congruence. The resulting term is not completely discrete because the temporal synchronization between concurrent hybrid processes needs to be maintained. In this paper, we (i) define the hybrid process calculus HCCS as a suitable minimalistic extension of CCS [18] , (ii) study its metatheory including an important connection between behavioural congruence and zenoness, (iii) state and prove properties that are required for a rigorous analysis of discretization, and (iv) apply our methodology to prove a hybrid tank system correct.

Keywords: Timed Process Calculi, Discretization, Hybrid Systems.

1 Introduction

A hybrid system is, from our perspective of modeling embedded systems, the combination of discrete algorithms with a continuous environment and consists of (i) a controller, modeled by a discrete algorithm, and (ii) an environment, modeled by differential equations (DEs). The controller is activated upon scheduled timeouts or events issued by the environment, it reads part of the status of the environment, applies a control strategy, and modifies the set of DEs of the environment to some extent (depending on the formalism) before hibernating.

The complexity of hybrid systems stems from the undecidability of the halting problem and the intractability of arbitrary DEs (both problems are inherited from their respective domain). Although semi-deciders (e.g., HyTech[15], HyperTech[16], HySAT[12]) and automatic theorem provers (e.g., for the differential dynamic logic $\mathsf{d}\mathcal{L}$ over nonparallel hybrid programs[20]) have been constructed, a theory for the rigorous verification of hybrid systems is called for: where automation fails, the software technician and a strong set of formal methodologies focused on user interaction for all reasonable selections of features of hybrid systems are indispensable.

In this paper we propose a behavior-preserving transformation (called *discretization*) to replace continuous steps by simple delays (for maintaining interprocess synchronization) and assignments (for preserving the observability of the environment by the controller[1]) to facilitate purely algebraic, thus rigorous

[1] Further discussed in the explanation of the rule CTermB on page 307.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 301–316, 2011.

(even formal), analysis. The transformation is **(i)** oblivious to an exchange of the expression languages that describe the DEs (up to solvability of the described DEs), **(ii)** an endomorphism preserving the term constructors, **(iii)** non-shallow in the sense that after discretization any reachable process has no derivation containing nontrivial CTs (continuous trajectories) (because only simple delays remain in the process after discretization), and **(iv)** not introducing infinite branching. The term discretization is reasonable, since, although the derived process contains realtime aspects, the complexity of its temporal behavior is negligible compared to hybrid systems.

The soundness of the approach (i.e., the behavior-preservation) is formalized by comparing the discretized term with its shallow (also termed semantic) discretization in any context using a reasonable notion of weak congruence. As we perform shallow abstraction by simple hiding of continuous behavior without any internal modification of the term, it is natural that the comparison with the shallow abstraction provides a rigorous and adequate measure for the goodness of syntactic discretization.

The discretization is valuable since **(i)** from our perspective, any formal verification would usually proceed along the lines of the construction of the syntactic discretization of this paper, that is, the elimination of the complex hybrid behaviour from the systems description to identify the durations and intermediate values of the trajectories observable from the hybrid system. Thus, the application of the discretization procedure is no additional overhead to the verification task. Furthermore, the major problem (aside from handling the discrete controller) consists in the exploration of possible continuous steps, which becomes trivial after discretization. Thus, our main contribution is the separation of the concerns (a) analysis of the CTs and (b) the correctness of the problem at hand. This separation can best be understood through the inspection of the proof of Theorem 6.1. **(ii)** the specification of reachability problems is facilitated. Without the presented methodology it is not possible to specify a reachability problem using reasonable structural notions of behavioral congruence, since (a) if shallow abstraction is used in the wrapper, then it is not possible to state properties about valuations intermediately traversed by the environment because it is precisely this information that is lost by the shallow abstraction and (b) if shallow abstraction is avoided in the wrapper, then the specification has to comprise a congruent temporal behavior that is also to be avoided for keeping the specification as simple as possible. One way to solve this problem is by using code annotations as follows: (a) the formal discretization derives the set of intermediately traversed valuations and (b) these valuations may be exposed to the environment by introducing an additional output prefix into the continuation of the CP (continuous prefix). Aside from the observability of this special action, the observational behavior of the closed loop system is unchanged. For example, this method facilitates the execution of a failure-exposing step whenever a bad state has been reached. *From the perspective of control engineers* the primary object of investigation is the continuous behavior of the system (as a set of traces) while the switching algorithms are understood as interruptions of the regular behavior. Our perspective is quite contrary: the CTs are intermediate calculations between two executions of the controller. *The structure of the paper* is as follows: The Sections 2, 3, and 4 introduce syntax, semantics, and the behavioral congruence of HCCS, Section 5 contains the discretization and our main properties thereof, Section 6 contains a case study exemplifying the methodology, and Section 7 concludes the paper with discussion about related and future work.

2 Syntax

In this section, we introduce the syntax of the process calculus HCCS, assuming that the reader is familiar with the widely used and well-known CCS [18] . The calculus is an extension of Value Passing CCS with value comparison ($[v = n] . P$) and calculation ($[\ \vec{x} \lhd f(\vec{v})\ ]\ .\ P$) by two operators for continuous behavior as follows. The hiding operator $(\varsigma\, x)\, P$ for the continuous variable x is neutral for discrete steps but removes x from any CT that is generated by P. Thus, the variable x is hidden and not observable from the context in which $(\varsigma\ x)\ P$ is executed; the other variables of a CT are not modified, thus the valuations of x may be deduced in some cases from the other observable valuations which are not removed from the CT. The operator describing CTs $[\,\alpha_1 \mid \phi \mid \alpha_2 \mid Q\,] . P$ contains five subterms: (i) the continuation term P is executed when the CP has reached completion without being aborted (the process P has access to the final valuation of the executed CT), (ii) the abortion process Q may communicate with a process evolving in parallel to Q' resulting in an abortion of the CP which is subsequently replaced by the Q', (iii) the assertion term α_1 defines the initial condition for the described CTs, (iv) the flow term ϕ defines the flow condition for the described CTs, and (v) the termination term α_2 holds precisely at the final time point of any derived CT. Thus, every generated CT is terminated and has therefore a finite length.

Definition 2.1 (Processes). *The sets of processes $\mathcal{P}$ $(P, Q, \dots)$ and summations $(M, N, \dots)$ are generated as follows:*

$$\begin{aligned} P &\triangleq P|P \,\Big|\, (\boldsymbol{\nu}\, a)\, P \,\Big|\, (\varsigma\, x)\, P \,\Big|\, \overline{a}\langle \vec{v} \rangle . P \,\Big|\, !\, P \,\Big|\, \mathbf{0} \,\Big|\, M \\ &\quad \Big|\, [\vec{x} \lhd f(\vec{v})] . P \,\Big|\, [v = n] . P \,\Big|\, [\alpha \mid \phi \mid \alpha \mid P] . P \\ M &\triangleq M + M \,\Big|\, a(\vec{x}) . P \end{aligned}$$

The set of pre-post-assertions $\mathcal{A}$ $(\alpha, \dots)$ contains finite sequences of equalities of the form $x = v$. The set of flows $\mathcal{F}$ $(\phi, \dots)$ contains finite sequences of equalities of the form $\dot{x} = v$. A variable x is defined with definition y iff the unique contained equality $x = z$ resp. $\dot{x} = z$ satisfies $y = z$[2]*. The set of values $\mathcal{V}$ $(r, \dots)$ contains real numbers and sets of real numbers. The set of variables $\mathcal{X}$ $(x, y, z, \dots)$ contains variables that may be substituted by values*[3]*. Finally, we*

[2] E.g., x is defined with definition y in $\dot{x} = y$ but not in $\dot{x}' = y$ (if $x \neq x'$) or $\dot{x} = y, \dot{x} = y'$ (if $y \neq y'$).

[3] Variables on the left side of $=$ in assertion and flow terms are termed *continuous variables* and are only affected by alpha renaming substitutions.

are using the set of extended values $\mathcal{V}^+ \triangleq \mathcal{V} \cup \mathcal{X}$ $(v, w, \dots)$, the set of names $\mathcal{N}$ $(a, b, c, \dots)$, natural numbers n, and total functions f on values.

For presentation purposes, the terms for pre-post-assertions and flows are quite restricted. However, more complex hybrid systems can be used by simply replacing the expression language for these terms, e.g., by allowing inequalities for assertions $x \leq 10$ and linear flow terms $\dot{x} = r \times y$.

Substitution of variables, and the sets of free names, bound names, variables, continuous variables, and recursion variables of terms are defined straightforward: $(\boldsymbol{\nu}\, a)\, P$ is a binder for names, $(\boldsymbol{\varsigma}\, x)\, P$ is a binder for continuous variables, and $[\, \vec{x} \lhd f(\vec{v})\,] . P$ and $a(\vec{x}) . P$ are binders for arbitrary variables.

Remark 2.2 (Notation). Aside from the standard notations for (i) omission of trailing $.\ \mathbf{0}$, (ii) omission of empty parameter lists, (iii) nested restrictions, and (iv) nested hiding, we use shortcuts for: (i) delayed abortable processes[4]: $[\ \delta \rhd Q\ ]\ .\ P \triangleq (\boldsymbol{\varsigma}\ t)\ [\ t = 0 \mid \dot{t} = 1 \mid t = \delta \mid Q\ ]\ .\ P$, (ii) delayed processes: $[\, \delta\,] . P \triangleq [\, \delta \rhd \mathbf{0}\,] . P$, (iii) full hiding $\curlywedge P \triangleq (\boldsymbol{\varsigma}\ \mathrm{fv}(P))\, P$ where $\mathrm{fv}(\cdot)$ is assumed to return a (finite) ordered sequence of the free continuous variables contained in P, (iv) complex comparisons $[\, f(\vec{x})\,] . P \triangleq [\, b \lhd f(\vec{x})\,] . [\, b = 1\,] . P$ where b is fresh, and (v) assuming recursion variables ranging over X, Y as process terms we can use recursion by encoding it using replication: $(\boldsymbol{\mu}\, X)\, P \triangleq (\boldsymbol{\nu}\, a)\ (\overline{a} \,|\, \boldsymbol{!}\, a . \{^{\overline{a}}/_{X}\}\, P)$

We conclude our definition of the syntactic elements of the calculus by straightforwardly defining process contexts, including prefixes and summations.

Definition 2.3 (Process Contexts). *The set of process contexts $\mathcal{C}$, ranging over C is generated by replacing a single subprocess of a process $P \in \mathcal{P}$ with $[\cdot]$.*

For example, the following important variants of control and systems can be modeled: (i) clock based systems (by using $[\ \delta\ ]\ .\ P$ and $[\ \delta \rhd Q\ ]\ .\ P$). (ii) event based systems (by expressing the event conditions in the termination condition of a CP), and (iii) systems where the evolution of a variable x depends on multiple components (by aborting and restarting the actual evolution of x with the updated flow whenever one of the influencing systems modifies its influence on the flow). Note, as for many CCS-like calculi, the HCCS processes contain the state of the whole system including the DEs and state of the enviroment of a modelled controller.

3 Semantics

In this section we introduce CTs along with essential combinators and comparators on them. In contrast to the usual habit for nonhybrid process calculi, some of our labels (the CTs) are not defined purely syntactically. We justify the usage of *nonsyntactic labels* (which seems unfavorable for syntax oriented formalism as this process calculus) for CTs by the additional expected overhead for the following definitions when using solely syntactical means.

[4] The variable t is assumed to be only used for simple delays.

The following rather generic definition of strings and some operations on them is required for our further presentation.

Definition 3.1 (Finite/Infinite Discrete/Continuous Strings). *For* $A \in \{\, \mathbf{N},\ \mathbf{R}^{\geq 0}\,\}$ *the set of all strings over* B *with index set* A *are given by* $A \twoheadrightarrow B \triangleq \{\, f : A \rightharpoonup B \mid \forall t \in \operatorname{supp}(f), t' < t\ .\ t' \in \operatorname{supp}(f)\,\}$ *(set of partial functions with leftclosed or empty support). For* $n \in \operatorname{supp}(f)$*,* $f(n)$ *is the element with index* n*. The duration* $\operatorname{dur}(f)$ *of* f *is the supremum of its support* $\operatorname{supp}(f)$ *over* $A \cup \{\infty\}$*. A string* f *is empty (denoted* $f = \lambda$*) iff* $\operatorname{supp}(f) = \emptyset$*. A string* f *is bounded iff* $\operatorname{supp}(f) \neq A$*. A nonempty string* $f : \mathbf{N} \twoheadrightarrow B$ *has a first element* $\operatorname{head}(f) = f(0)$*. A bounded nonempty string* $f : \mathbf{N} \twoheadrightarrow B$ *has a tail string* $\operatorname{tail}(f)(n) = f(n+1)$ *(for* $n < \max(\operatorname{supp}(f))$*), and a last element* $\operatorname{last}(f) = f(\max(\operatorname{supp}(f)))$*. If* $f_1 : \mathbf{N} \twoheadrightarrow B$ *is a bounded string and* $f_2 : \mathbf{N} \twoheadrightarrow B$ *is a string, then their ordinary sequential concatenation is* $f_1 f_2 = h : \mathbf{N} \twoheadrightarrow B$ *iff* $f_1 = f_2 = h = \lambda$ *or*

$$h(n) = \begin{cases} f_2(n) & ,\ f_1 = \lambda \neq f_2 \wedge n \leq \max(\operatorname{supp}(f_2)) \\ f_1(n) & ,\ f_1 \neq \lambda = f_2 \wedge n \leq \max(\operatorname{supp}(f_1)) \\ f_1(n) & ,\ f_1, f_2 \neq \lambda \wedge n \leq \max(\operatorname{supp}(f_1)) \\ f_2(n - \max(\operatorname{supp}(f_1))) - 1) & ,\ \begin{cases} \quad f_1, f_2 \neq \lambda \\ \wedge \max(\operatorname{supp}(f_1)) < n \\ \wedge\, n \leq \max(\operatorname{supp}(f_2)) \end{cases} \end{cases}$$

and the non duplicating sequential concatenation $f_1 \odot f_2 = g : \mathbf{N} \twoheadrightarrow B$ *iff where*

$$g = \begin{cases} f_1 \operatorname{tail}(f_2) & ,\ \operatorname{last}(f_1) = \operatorname{first}(f_2) \\ f_1 f_2 & ,\ \textit{else} \end{cases}$$

A CT describes the evolution of (some of) the continuous variables over (some) fragment of time[5]. Since continuous variables may jump during instantaneous switching (successive CTs are concatenated[6]), a single variable may have multiple values at a single point in time; as usual, all such time-identical values are represented as a trace of countable length.

Definition 3.2 (Continuous Trajectory). *A map* $\xi : \mathbf{R}^{\geq 0} \twoheadrightarrow (\mathcal{X} \rightarrow (\mathbf{N} \twoheadrightarrow \mathbf{R}))$ *is a CT iff (i)* $\xi \neq \lambda$*, (ii)* $\sup^{\mathcal{X}}(\xi) \triangleq \{\, x \in \mathcal{X} \mid \exists t \in \operatorname{supp}(\xi)\ .\ \xi(t)(x) \neq \lambda\,\}$ *(the set of variables that* ξ *mentions) is finite, and (iii) for any* $t \in \operatorname{supp}(\xi)$*:* $\operatorname{dur}(\xi(t)(x)) \neq \infty$*.* Ξ *is the set of all CTs.*

We continue with some basic definitions about CTs.

Definition 3.3 (Empty CT). *The set* $\Xi_\emptyset = \{\, \xi \in \Xi \mid \sup^{\mathcal{X}}(\xi) = \emptyset\,\}$ *contains all variable free CTs.* $\xi^r_\emptyset \in \Xi_\emptyset$ *is the unique CT with* $\operatorname{dur}(\xi) = r$ *and greatest support.*

[5] A single step generates CTs of finite duration but sequential CTs may add up to infinite duration.

[6] Concatenation takes place in the 4th rule in Figure 2.

Definition 3.4 (Final Valuation of CTs). *The final valuation* $\mathrm{final}(\xi) : \mathcal{X} \rightharpoonup \mathbf{R}$ *of a CT* ξ *satisfies (i)* $\mathrm{supp}(\mathrm{final}(\xi)) = \sup^{\mathcal{X}}(\xi)$ *and (ii) for any* $x \in \sup^{\mathcal{X}}(\xi)$*:* $\mathrm{final}(\xi)(x) = \mathrm{last}(\xi(t)(x)) = \xi(t)(x)(\max(\mathrm{supp}(\xi(t)(x))))$ *s.t. there is no* $t' > t$ *satisfying* $\xi(t')(x) \neq \lambda$*.*

Remark 3.5. The t in the last definition is not necessarily identical for each x: A variable has a last value even if it is not used throughout the evolution of the system but only duration a certain period.

Sequential composition of CTs is straightforward except for the join point where for non-jumping continuous variables a replication of the last value is removed to enable the invisibility of interruptions after concatenation of CTs.

Definition 3.6 (Sequential Composition of CTs). *Let* ξ_1 *and* ξ_2 *be two CTs. Their sequential composition* $\xi_1;\xi_2$ *is the CT* ξ *iff (i)* $\mathrm{dur}(\xi) = \mathrm{dur}(\xi_1) + \mathrm{dur}(\xi_2)$*, (ii)* $\mathrm{dur}(\xi_1) \neq \infty$*, (iii)* $\xi(t) = \xi_1(t)$*, if* $t < \mathrm{dur}(\xi_1)$*, (iv)* $\xi(t) = \xi_2(t - \mathrm{dur}(\xi_1))$*, if* $t > \mathrm{dur}(\xi_1)$*, and (v) for* $t = \mathrm{dur}(\xi_1)$ *and any x:* $\xi(t)(x) = \xi_1(t)(x) \odot \xi_2(0)(x)$*. A CT* ξ_1 *is a temporal extension of a CT* ξ_2 *(written* $\xi_1 \gg \xi_2$*) iff there is a CT* ξ_3*, s.t.,* $\xi_1 = \xi_2;\xi_3$*.*

Remark 3.7. If the first CT has a nonterminating discrete behaviour (livelock or zeno behaviour (cf. Definition 4.5)) at the last point in time (formally $\xi_1(t)(x)$ is not finite), then the sequential composition $\xi_1;\xi_2$ is undefined.

Parallel composition of CTs can be defined to combine CTs according to complex combinators as in HYPE [13]. Since these combinators are the subject of our analysis (and are therefore supposed to be expressed within the terms) we are simply using a simple union operation as composition operator for non-conflicting CTs.

Definition 3.8 (Parallel Composition of CTs). *Let* ξ_1 *and* ξ_2 *be two CTs. Their parallel composition* $\xi_1 \cup \xi_2$ *is* ξ *iff for* $t \in \mathrm{supp}(\xi)$ *and variables* $x \in \mathcal{X}$ *it holds that (i)* $\mathrm{dur}(\xi) = \mathrm{dur}(\xi_1) = \mathrm{dur}(\xi_2)$*, (ii)* $\xi(t)(x) = \xi_1(t)(x)$*, if* $\xi_1(t)(x) \neq \lambda$ *and otherwise* $\xi(t)(x) = \xi_2(t)(x)$*, and (iii)* $\xi_1(t)(x) \neq \xi_2(t)(x)$ *then* $\xi_1(t)(x) = \lambda$ *or* $\xi_2(t)(x) = \lambda$*. A CT* ξ_1 *is a variable extension of a CT* ξ_2 *(written* $\xi_1 > \xi_2$*) iff* $\xi_1 \supseteq \xi_2$*.*

Remark 3.9. The parallel composition is associative, commutative, and for any CT ξ the (empty) CT $\xi_\emptyset^{\mathrm{dur}(\xi)}$ is neutral. The sequential composition is associative with neutral element $\xi_\emptyset^0$ (given the first CT has finite duration).

We now introduce maps for the translation of assertion ($\mathcal{A}$) and flow ($\mathcal{F}$) predicates into set theoretic valuations and CTs.

Definition 3.10 (Semantics for Assertions, Flows, and CPs). *The map* $\langle\!\langle\ \rangle\!\rangle : \mathcal{A} \cup \mathcal{F} \rightharpoonup (\mathcal{X} \rightharpoonup \mathbf{R})$ *returns the set theoretic valuations for* $\mathcal{A}$ *and* $\mathcal{F}$ *terms*[7]*. The map* $[\![\ ]\!] : \mathcal{A} \times \mathcal{F} \times \mathcal{A} \rightarrow 2^{\Xi}$ *returns the set theoretic CTs described by a CP. Their concrete definition is straightforward.*

[7] Partiality is due to possibly contradicting subterms.

We are now all set to define the labels used in the semantics.

Definition 3.11 (Labels). *$\mathcal{L}_{\mathrm{C}} = \varXi$ is the set of (unannotated) CTs ξ. $\mathcal{L}_{\mathrm{C}}^{\mathrm{A}}$ contains for $\xi \in \varXi$ the annotated CTs $0\natural\xi$ and $1\natural\xi$. The set $\mathcal{L}_{\mathrm{D}}$, ranged over by μ, contains the labels for input $a(\vec{v})$, output $\overline{a}\langle\vec{v}\rangle$, and unspecified interaction τ. The set $\mathcal{L}$, ranged over by κ, is the union of $\mathcal{L}_{\mathrm{C}}$ and $\mathcal{L}_{\mathrm{D}}$.*

In the maximal progress semantics of this calculus, the duration of a common CT of multiple parallel processes is determined by the quickest process (i.e., the process whose maximal active continuous step has the shortest duration among all contained processes.) The annotation distinguishes between passive steps that are restricted by some parallel process (annotation 0) and active steps which can not be extended (annotation 1).

Finally, we present the definition of the LTS of the calculus.

Definition 3.12 (Labelled Transition System for HCCS). *The semantics of* HCCS *is given by the LTS $(\mathcal{P}, \mathcal{L}, \longrightarrow)$ where $\longrightarrow \triangleq (\rightsquigarrow \cup \leadsto) \subset \mathcal{P} \times \mathcal{L} \times \mathcal{P}$, $\rightsquigarrow \subset \mathcal{P} \times \mathcal{L}_{\mathrm{C}} \times \mathcal{P}$, $\leadsto \subset \mathcal{P} \times \mathcal{L}_{\mathrm{D}} \times \mathcal{P}$, $\dashrightarrow \subset \mathcal{P} \times \mathcal{L}_{\mathrm{C}}^{\mathrm{A}} \times \mathcal{P}$ are the smallest relations satisfying the rules in Figure 1.*

We omit any detailed discussion on the standard rules of the Value Passing CCS fragment (rules Alpha through Asgn). Note that the semantics is "early", which implies that closed terms always evolve into closed terms and that the greedy construction of continuous steps ensures that only the early input prevents the transition system of being finitely branching. The other rules behave as follows: **(i)** CNeut: every process that can not perform some discrete or active continuous step allows time to progress without restricting the possible behavior, **(ii)** Conv: only steps of active CTs are transferred into the LTS, **(iii)** CAbort: the abortion of the continuous evolution is possible when the abortion process Q can communicate with the environment, however, zero length CTs can not be aborted, **(iv)** CAbortP: the abortion process may perform internal computations, **(v)** CPass: a process can perform a passive step, if it can be extended to an active step where the CT ξ of the passive step is created solely by the CP in front of P, **(vi)** CTermA: multiple CPs may be consumed at once, CTs of subsequently consumed CP are composed sequentially. The side-condition states that ξ completes the CP in front of P, **(vii)** CTermB: the recursive greedy construction by sequential composition of active CTs terminates if the subsequent term can execute only the passive step generated by the CNeut rule where the substitutions on P model that the controller measures the environment upon activation, **(viii)** CJoin: two parallel terms may execute a CT if their individual CTs can be composed according to Definition 3.8 and no τ step is immediately available (it is certainly acceptable to have blocked inputs and outputs). The resulting CTs are active iff at least one of the two CTs ξ_1, ξ_2 is active, i.e., if $\max(k_1, k_2) = 1$. The side condition requires that the CT of each passive step can be extended to a CT of some active step, **(ix)** CRes1: abusing notation all traces of the variable c (for any point in time) are removed from ξ, and **(x)** CRes2: continuous restriction is neutral for discrete steps.

$$\textsc{Alpha}\ \frac{Q \xrightarrow{\kappa} Q'}{P \xrightarrow{\kappa} Q'}\ P =_\alpha Q \qquad \textsc{ASum1}\ \frac{P \overset{\mu}{\rightsquigarrow} P'}{P + Q \overset{\mu}{\rightsquigarrow} P'} \qquad \textsc{Inp}\ \frac{}{a(\vec{x})\,.\,P \overset{a(\vec{v})}{\rightsquigarrow} \{\vec{v}/\vec{x}\}\,P}$$

$$\textsc{Out}\ \frac{}{\overline{a}\langle\vec{v}\rangle\,.\,P \overset{\overline{a}\langle\vec{v}\rangle}{\rightsquigarrow} P}\ \{\,\vec{v}\,\} \subseteq \mathcal{V} \qquad \textsc{Rep}\ \frac{P\,|\,!P \overset{\mu}{\rightsquigarrow} P'}{!P \overset{\mu}{\rightsquigarrow} P'}$$

$$\textsc{ARes}\ \frac{P \xrightarrow{\kappa} P'}{(\boldsymbol{\nu} a)\,P \xrightarrow{\kappa} (\boldsymbol{\nu} a)\,P'}\ a \notin \mathrm{names}(\kappa) \qquad \textsc{APar1}\ \frac{P \overset{\mu}{\rightsquigarrow} P'}{P\,|\,Q \overset{\mu}{\rightsquigarrow} P'\,|\,Q}$$

$$\textsc{Com1}\ \frac{P \overset{a(\vec{v})}{\rightsquigarrow} P' \qquad Q \overset{\overline{a}\langle\vec{v}\rangle}{\rightsquigarrow} Q'}{P\,|\,Q \overset{\tau}{\rightsquigarrow} P'\,|\,Q'} \qquad \textsc{Cond}\ \frac{P \overset{\mu}{\rightsquigarrow} P'}{[v = n]\,.\,P \overset{\mu}{\rightsquigarrow} P'}\ v = n$$

$$\textsc{Asgn}\ \frac{\{\vec{w}/\vec{x}\}\,P \overset{\mu}{\rightsquigarrow} P'}{[\vec{x}\lhd f(\vec{v})]\,.\,P \overset{\mu}{\rightsquigarrow} P'}\ \begin{cases} f(\vec{v}) = \vec{w} \\ \{\,\vec{v}\,\} \subset \mathcal{V} \end{cases} \qquad \textsc{CNeut}\ \frac{\forall \xi'\,.\,P \overset{1\natural\xi'}{\not\dashrightarrow} \qquad P \overset{\tau}{\not\rightsquigarrow}}{P \overset{0\natural\xi}{\dashrightarrow} P}\ \xi \in \varXi_\emptyset$$

$$\textsc{Conv}\ \frac{P \overset{1\natural\xi}{\dashrightarrow} P'}{P \overset{\xi}{\rightsquigarrow} P'}$$

$$\textsc{CAbort}\ \frac{\langle\!\langle\, \alpha_1 \,\rangle\!\rangle Q \overset{\mu}{\rightsquigarrow} Q'}{[\alpha_1 \mid \phi \mid \alpha_2 \mid Q]\,.P \overset{\mu}{\rightsquigarrow} Q'}\ \exists \xi \in [\![\, \alpha_1, \phi, \alpha_2 \,]\!]\,.\ \mathrm{dur}(\xi) \neq 0,\ \mu \neq \tau$$

$$\textsc{CAbortP}\ \frac{\langle\!\langle\, \alpha_1 \,\rangle\!\rangle Q \overset{\tau}{\rightsquigarrow} Q'}{[\alpha_1 \mid \phi \mid \alpha_2 \mid Q]\,.P \overset{\tau}{\rightsquigarrow} [\alpha_1 \mid \phi \mid \alpha_2 \mid Q']\,.P}\ \exists \xi \in [\![\, \alpha_1, \phi, \alpha_2 \,]\!]\,.\ \mathrm{dur}(\xi) \neq 0$$

$$\textsc{CPass}\ \frac{[\alpha_1 \mid \phi \mid \alpha_2 \mid Q]\,.\,P \overset{1\natural\xi''}{\dashrightarrow}}{[\alpha_1 \mid \phi \mid \alpha_2 \mid Q]\,.\,P \overset{0\natural\xi}{\dashrightarrow} [\alpha_1' \mid \phi \mid \alpha_2 \mid Q]\,.\,P}\ \begin{cases} \xi' \in [\![\, \alpha_1, \phi, \alpha_2 \,]\!] \\ \xi \ll \xi' \ll \xi'' \\ \langle\!\langle\, \alpha_1' \,\rangle\!\rangle = \mathrm{final}(\xi) \end{cases}$$

$$\textsc{CTermA}\ \frac{\{\mathrm{final}(\xi)(\vec{x})/\vec{x}\}\,P \overset{k\natural\xi'}{\dashrightarrow} P'}{[\alpha_1 \mid \phi \mid \alpha_2 \mid Q]\,.\,P \overset{k\natural\xi;\xi'}{\dashrightarrow} P'}\ \begin{cases} \xi \in [\![\, \alpha_1, \phi, \alpha_2 \,]\!] \\ \vec{x} = \sup^{\mathcal{X}}(\xi) \end{cases}$$

$$\textsc{CTermB}\ \frac{\{\mathrm{final}(\xi)(\vec{x})/\vec{x}\}\,P \overset{k\natural\xi'}{\not\dashrightarrow} P'}{[\alpha_1 \mid \phi \mid \alpha_2 \mid Q]\,.\,P \overset{1\natural\xi}{\dashrightarrow} \{\mathrm{final}(\xi)(\vec{x})/\vec{x}\}\,P}\ \begin{cases} \xi \in [\![\, \alpha_1, \phi, \alpha_2 \,]\!] \\ \vec{x} = \sup^{\mathcal{X}}(\xi) \\ k \neq 0 \vee \xi' \notin \varXi_\emptyset \end{cases}$$

$$\textsc{CJoin}\ \frac{P \overset{k_1\natural\xi_1}{\dashrightarrow} P' \qquad Q \overset{k_2\natural\xi_2}{\dashrightarrow} Q' \qquad P\,|\,Q \overset{\tau}{\not\rightsquigarrow}}{P\,|\,Q \overset{\max(k_1,k_2)\natural\xi_1\cup\xi_2}{\dashrightarrow} P'\,|\,Q'}\ \max(k_1, k_2) = 1 \vee P\,|\,Q \overset{1\natural\xi}{\dashrightarrow}$$

$$\textsc{CRes1}\ \frac{P \overset{k\natural\xi}{\dashrightarrow} P'}{(\varsigma\, c)\,P \overset{k\natural\xi[c\mapsto\textsc{undef}]}{\dashrightarrow} (\varsigma\, c)\,P'} \qquad \textsc{CRes2}\ \frac{P \overset{\mu}{\rightsquigarrow} P'}{(\varsigma\, c)\,P \overset{\mu}{\rightsquigarrow} (\varsigma\, c)\,P'}$$

Fig. 1. Rules for the construction of the strong LTS

$$\frac{P \overset{\mu}{\rightsquigarrow} Q}{P \overset{\mu}{\Longrightarrow} Q} \qquad \frac{P \overset{k\natural\xi}{\dashrightarrow} Q}{P \overset{\xi}{\Longrightarrow} Q} \qquad \frac{P \multimap_\xi Q}{P \overset{\xi}{\Longrightarrow} Q} \qquad \frac{P \overset{\xi_1}{\Longrightarrow} \overset{\xi_2}{\Longrightarrow} Q}{P \overset{\xi_1;\xi_2}{\Longrightarrow} Q} \qquad \frac{}{P \overset{\tau}{\Longrightarrow} P} \qquad \frac{P \overset{\tau}{\Longrightarrow} \overset{\kappa}{\Longrightarrow} \overset{\tau}{\Longrightarrow} Q}{P \overset{\kappa}{\Longrightarrow} Q}$$

Fig. 2. Rules for the construction of the weak LTS

4 Behavioral Congruence

This section covers a definition of weak congruence capable of handling continuous evolutions and our congruence theorem.

The definition of weak steps comprises CTs from active steps, passive steps (for the simulation of active steps), and CTs constructed by infinite sequential composition of CTs leading to the unique (up to strong bisimilarity) limit process. The intermediate processes need to be reduced in size[8] using strong bisimilarity to reach a finite limit process.

Definition 4.1 (Weak Step). *Let* $\Longrightarrow\ : \mathcal{P} \times \mathcal{L} \times \mathcal{P}$ *be the smallest relation satisfying the rules in Figure 2.*

Definition 4.2 ($\mathcal{R}$ (Bi)Simulation). *A relation* $R \subseteq \mathcal{P} \times \mathcal{P}$ *is a* $\mathcal{R}$ *simulation iff* $P\ R\ Q$ *implies that*

- *if* $P \overset{\mu}{\rightsquigarrow} P'$, *then there is* Q' *with* $(Q, \mu, Q') \in \mathcal{R}$ *and* $P'\ R\ Q'$.
- *if* $P \overset{\xi}{\rightsquigarrow} P'$, *then there are* Q' *and* $\xi' > \xi$ *with* $(Q, \xi', Q') \in \mathcal{R}$ *and* $P'\ R\ Q'$.

R is a strong (weak) simulation iff R is a $\longrightarrow$ *(*$\Longrightarrow$*) simulation.*
R is a strong (weak) bisimulation iff R and R^{-1} are strong (weak) simulations.
Strong (weak) bisimilarity $\sim$ ($\approx$) is the largest strong (weak) bisimulation.

Definition 4.3 (Limit Process). *The limit process of P is* $\{\vec{v}_l/\vec{x}\}\,Q$ *with limit CT ξ (denoted* $P \multimap_{\xi} \{\vec{v}_l/\vec{x}\}\,Q$*) iff (i) Q is value free (i.e., no element of $\mathcal{V}$ is contained), (ii)* $(\xi_i \in \Xi)_{i \in \mathbf{N}}$ *are CTs, (iii)* $(P_i \in \mathcal{P})_{i \in \mathbf{N}}$ *are value free processes, (iv)* $(\vec{v}_i)_{i \in \mathbf{N}}$ *are vectors of values converging pointwise to* $\vec{v}_l$, *(v)* $\{\vec{v}_0/\vec{x}\}\,P_0 = P$, *(vi)* $\forall n \in \mathbf{N}\ .\ \{\vec{v}_n/\vec{x}\}\,P_n \overset{\xi_n}{\Longrightarrow} \sim \{\vec{v}_{n+1}/\vec{x}\}\,P_{n+1}$, *and (vii)* $\xi = \xi_0; \xi_1; \ldots$.

We continue with the definition of weak steps which makes τ steps unobservable.

Note that non-trivial *open* terms[9] may not have steps themselves and thus be (weak) bisimilar to $\mathbf{0}$. When plugged into a context where the hole occurs in the scope of binders, previously impossible steps become enabled. Thus, as usual, weak bisimilarity is no full congruence. To achieve a full congruence property, we refine weak bisimilarity by requiring closure under arbitrary substitutions.

Definition 4.4 (Weak Congruence). *The relation* $\approx^{\mathrm{c}} \subseteq \mathcal{P} \times \mathcal{P}$ *is defined by:* $P \approx^{\mathrm{c}} Q$ *iff for any substitution σ is holds that* $\sigma P \approx \sigma Q$.

Since discrete steps are instantaneous and the duration of continuous steps has no lower bound (minimal dwell time) HCCS produces the same unrealistic modeling artifact as in many other timed formalisms: zenoness. As usual, we consider a process P to be zeno iff P can do an infinite number of steps in finite time.

Definition 4.5 (Zeno Process). *A process* $P \in \mathcal{P}$ *is zeno iff there are two sequences* $\eta_1 : \mathbf{N} \rightarrow \mathcal{L}$ *and* $\eta_2 : \mathbf{N} \rightarrow \mathcal{P}$ *s.t. (i) the sequences model an evolution according to the* HCCS *semantics:* $\forall n \in \mathbf{N}\ .\ \eta_2(n) \xrightarrow{\eta_1(n)} \eta_2(n+1)$,

[8] Garbage collection of dead code such as $[0 = 1].P \sim P$ and $P\,|\,\mathbf{0} \sim P$.

[9] A term is open (resp. closed) iff it has (resp. has not) free variables (which could be replaced via an input).

(ii) the evolution has P as its origin: $\eta_2(0) = P$, and (iii) time converges: $\sum_{n \in \mathbf{N}} \mathrm{dur}(\eta_1(n)) \neq \infty$ (where for any discrete label μ: $\mathrm{dur}(\mu) = 0$).

The definition of weak congruence is inadequate for zeno processes because it lacks the means (i) to detect limit points (points in time to which time converges) and (ii) to analyze the processes' behavior after any reachable limit point.

Example 4.6 (Zenoness Breaks Congruence). Consider the zeno processes $P_0 \triangleq (\boldsymbol{\nu}\, a)\,(\boldsymbol{!}\, a \,|\, \boldsymbol{!}\,\overline{a})$ and $P_1 \triangleq (\boldsymbol{\nu}\, a)\,(\overline{a}\langle 1\rangle \,|\, \boldsymbol{!}\, a(x)\,.\,[x \lhd x/2]\,.\,[x]\,.\,([x = 0]\,.\,\overline{b} \,|\, \overline{a}\langle x\rangle))$ with zeno behavior. The weak congruences $P_0 \approx^{\mathrm{c}} \mathbf{0}$ and $P_1 \approx^{\mathrm{c}} \mathbf{0}$ are not preserved by the context $C \triangleq [\,1\,]\,.\,\mathbf{0} \,|\, [\cdot]$. The process $C[\mathbf{0}]$ has the step labeled with $\xi^1_\emptyset$. This step can not be simulated by $C[P_0]$ because the process P_0 is not capable of doing any temporal step at all. The situation is different for the process $C[P_1]$ which can simulate the step but the resulting term has a $\overline{b}\langle\rangle$ step which can not be simulated by the continuation of $C[\mathbf{0}]$ after executing the $\xi^1_\emptyset$ step.

This example demonstrates that the absence of zenoness (including divergence) must be verified separately to ensure the adequacy of $\approx^{\mathrm{c}}$. Nevertheless, excluding zeno processes is sufficient to derive the following congruence property.

Theorem 4.7 (Weak Bisimilarity is a Congruence). *For all contexts $C \in \mathcal{C}$ and all nonzeno processes $P, Q \in \mathcal{P}$, if $P \approx^{\mathrm{c}} Q$, then $C[P] \approx^{\mathrm{c}} C[Q]$.*

Proof (Sketch). We exhibit a weak congruence containing the pair $(C[P], C[Q])$: $\dot{\approx} \triangleq \{\, (\, C[P_1],\ C[P_2]\,)\ \mid\ \text{nonzeno } P_1, P_2 \in \mathcal{P},\ C \in \mathcal{C},\ P_1 \approx^{\mathrm{c}} P_2 \,\} \cup \approx^{\mathrm{c}}$ (on open terms) and proof that it is contained in $\approx^{\mathrm{c}}$. The proof proceeds by case analysis on the context C. The most demanding cases are (as expected) the continuous steps with the contexts $P \,|\, [\cdot]$ and $[\,\alpha_1 \mid \phi \mid \alpha_2 \mid Q\,]\,.\,[\cdot]$. □

Sufficient (at best structural) conditions for nonzenoness are called for.

Proposition 4.8 (Nonzenoness). *A process P is nonzeno, if in every of its executions (i) infinitely many CTs with nonzero-duration occur and (ii) finitely many values v are communicated.*

Proposition 4.8 is satisfactory for P_0 and P_1 from Example 4.6 and the system in the Toy Example in Section 6. The formal (in depth) treatment of this aspect is beyond the scope of this paper.

5 Syntactic Discretization

Discretization simplifies the continuous behavior while preserving the observable behavior to facilitate less complex proofs of relevant statements about the embedded system or its controller. Shallow discretization, implemented in HCCS via the operator $\measuredangle P$, does not modify the internals of P, which therefore retains its full complexity. Therefore shallow discretization is useless for verification purposes. Nevertheless, we consider it to be an adequate benchmark for a syntactic discretization: both notions should coincide.

Example 5.1 (Discretization in a Small Action). Consider the hybrid process H and its syntactic discretization D (where obvious simplifications have been executed).

$$H \triangleq (\boldsymbol{\nu}\, a, e)\,(\varsigma\, y) \begin{cases} \;\overline{a}\langle 5\rangle\,.\,[1]\,.\,\overline{e} \\ \mid a(x)\,.\,[y = 1 \mid \dot{y} = 2 \mid y = x \mid e\,.\,\overline{ob}\langle y\rangle\,]\,.\,\overline{ob}\langle y\rangle \end{cases}$$

$$D \triangleq (\boldsymbol{\nu}\, a, e)\,(\varsigma\, y) \begin{cases} \;\overline{a}\langle 5\rangle\,.\,[1]\,.\,\overline{e} \\ \mid a(x)\,.\,[d \lhd (x-1)/2]\,.\,[d \rhd [y \lhd 2 \times t + 1]\,.\,e\,.\,\overline{ob}\langle y\rangle\,]\,.\,\overline{ob}\langle x\rangle \end{cases}$$

This example demonstrates how free variables in the CP and abortions are handled. Furthermore, it demonstrates how the obvious reasoning steps about H are implemented by applying the discretization, i.e., verification of H would as well require the calculation of the durations and final valuations.

After this introductory example, we give the definition of discretization which replaces each CP with a temporal delay to preserve the temporal behaviour and an assignment to retain the final values of the CT of the CP.

Definition 5.2 (Discretization). *We define an operation* $\{\!\{\ \}\!\} : \mathcal{P} \rightharpoonup \mathcal{P}$ *that abstracts a (possibly continuous) process to a simply timed process. If two parallel processes share common continuous variables the discretization is undefined. The definition is homomorphic for all operations except CP, e.g.,* $\{\!\{\ (\boldsymbol{\nu}\, a)\, P\ \}\!\} \triangleq (\boldsymbol{\nu}\, a)\{\!\{\ P\ \}\!\}$. *The operation is now defined for the CP*

$$\begin{array}{rl}
\multicolumn{2}{l}{\{\!\{\ [\alpha_1 \mid \phi \mid \alpha_2 \mid Q]\,.\,P\ \}\!\} \triangleq} \\
\textit{compute delay} & [d \lhd f(\vec{v}_1, \vec{v}_2, \vec{v}_3)]\,. \\
\textit{delay} & [d \rhd \\
\textit{flow abstraction} & \quad [\vec{x}, \vec{x}^{\circ} \lhd g(\vec{v}_1, \vec{v}_2, t)]\,. \\
\textit{abortion} & \quad \{\!\{\ Q\ \}\!\}]\,. \\
\textit{flow abstraction} & [\vec{x}, \vec{x}^{\circ} \lhd g(\vec{v}_1, \vec{v}_2, d)]\,. \\
\textit{continuation} & \{\!\{\ P\ \}\!\}
\end{array}$$

where ***(i)*** variable for the duration: *the fresh variable d is for storing the duration of the CT,* ***(ii)*** calculation of the duration: *the function* $f : \mathbf{R}^* \to \mathbf{R}$ *computes the minimal duration for which the abortion criterion α_2 is satisfied:* $f(\vec{v}_i, \vec{v}_r, \vec{v}_t) \triangleq \min(\{\ \mathrm{dur}(\vec{v}_{i,k}, \vec{v}_{r,k}, \vec{v}_{t,k}) \mid 1 \leq k \leq |\vec{v}_i|\ \} \cup \{\ -1 \mid |\vec{v}_i| = 0\ \})$ *where (a)* $\mathrm{dur}(i, r, t) = (t - i)/r$ *if $r > 0$ and $t \geq i$ or $r < 0$ and $i \geq t$, (b)* $\mathrm{dur}(i, r, t) = 0$ *if $r = 0$ and $i = t$, and (c)* $\mathrm{dur}(i, r, t) = -1$ *for all other cases,* ***(iii)*** flow variables: *$\vec{x}$ contains precisely the variables defined in α_1 and ϕ,* ***(iv)*** duration determining parameters: *$\vec{v}_i$ ($\vec{v}_r$, $\vec{v}_t$) contains for each $x \in \vec{x}$ for which x is defined in α_2 the definition of x contained in α_1 (ϕ, α_2), and* ***(v)*** computation of postflow values: *$\vec{x}^{\circ}$ is obtained from $\vec{x}$ by adding $^{\circ}$ to each contained variable;* $g : \mathbf{R}^* \to \mathbf{R}^* \times 2^{\mathbf{R}^*}$ *computes the final value and the set of intermediate values for the variables that participate in the flow.* $g(\vec{v}_i, \vec{v}_r, d)$ *computes (a) for the k-th variable x from $\vec{x}$ the value $\vec{v}_{i,k} + d \times \vec{v}_{r,k}$ and (b) for the k-th variable x° from $\vec{x}^{\circ}$ the value* $[\vec{v}_{i,k}, \vec{v}_{i,k} + d \times \vec{v}_{r,k}] \cup [\vec{v}_{i,k} + d \times \vec{v}_{r,k}, \vec{v}_{i,k}]$.

Obviously the duration of a continuous step may depend on free variables whose future replacement can not be determined statically (by the undecidability of the halting problem) as in Example 5.1. We term our syntactic discretization complete since both functions f and g can be given in the explicit form of Definition 5.2. Complex classes of DEs (i.e., classes in which DEs can not be solved automatically) can be used using a different expression language. For such expression languages the completeness of the discretization procedure is unachievable by the intractability of the expressed classes of DEs. However, even a (partial) discretization using some incomplete discretization procedure could still simplify the process. Even for these expression languages where the derivation of durations can not be automated the duration can be approximated or manually derived. Verified discretization procedures for more complex DEs are deferred to future work. The partiality of our definition is solely due to parallel processes with reachable conflicting CTs (i.e., reachable temporal deadlocks) which are certainly implementation faults to be excluded. Our approach for prohibiting parallel processes with intersecting continuous variables is an overapproximation of the set of deadlocking processes. The condition is statically decidable and, as we believe, no restriction for real world applications. It merely enforces a compositional implementation of the system using explicit rather than implicit combination (as in [13]) of multiple influences.

Discretization restricts the observability to discrete steps and durations of continuous steps. As already explained in the introduction, the abstracted information can be recovered by standard code annotation techniques: the additional variable y° for storing the intermediate values may be used in a unobstrusive output in the continuation on a fresh name n by replacing the discretization $\{\!\!\{\, P \,\}\!\!\}$ of the continuation of the CP by $\{\!\!\{\, P \,\}\!\!\} | \overline{a}\langle y^\circ \rangle$ in the resulting process.

The following notion of continuous discrete equivalence relates processes P and Q iff their semantic discretization cannot be distinguished by any context w.r.t. weak congruence.

Definition 5.3 (CD-equivalence)
$P \simeq_{\mathrm{CD}} Q$ iff for all contexts $C \in \mathcal{C}$: $C[\reflectbox{$\lambda$} P] \approx^{\mathrm{c}} C[\reflectbox{λ} Q]$.

The following theorem contains the major property of this section and states that syntactical discretization coincides with the semantical abstraction w.r.t. weak congruence.

Theorem 5.4 (Syntactic Discretization is Sound).
For all nonzeno $P \in \mathcal{P}$: $\{\!\!\{\, P \,\}\!\!\} \approx^{\mathrm{c}} \reflectbox{λ} P$.

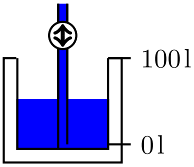
Proof *(Sketch)*. We exhibit a weak congruence (on open terms) relating syntactic and semantic discretization: $C = \{\, (\{\!\!\{\, P \,\}\!\!\}, \reflectbox{λ} P) \mid \text{nonzeno } P \in \mathcal{P} \,\} \cup \approx^{\mathrm{c}}$. The proof of $C \subseteq \approx^{\mathrm{c}}$ is straightforward except for the case of CP. There, it is obvious that both terms have two types of steps (abortions and steps completing the simple delay or the CP). By construction and careful analysis, the bisimulation conditions are verified. □

The intuition that syntactic discretization preserves behavior is captured as follows.

Corollary 5.5 (Discretization is Sound).
For all nonzeno $P \in \mathcal{P}$: $P \simeq_{\mathrm{CD}} \{\!\{ P \}\!\}$.

Proof *(Sketch).* By the congruence Theorem 4.7 and Theorem 5.4. □

6 Toy Example



A tank with 100 l volume has a pump that either removes 10 l/s from the tank or inserts 10 l/s into the tank. The goal of the controller is that the tank is at about 50 l, is never full/empty, and the valve is not switching too often. The controller polls the tank with period 1 s.

We proceed as follows: **(a)** we give a system process H: H generates an interrupt for the controller, sending the current status of the tank; then it receives the control signal from the controller; then 1 s elapses before the loop is restarted. **(b)** we give a controller process C: C receives the current status of the tank; according to the control strategy a control signal is sent to the system; finally the loop is restarted. **(c)** we give the nonreachability specification as a discrete (simply timed) process S: after a finite sequence of high/low signals the valve is switched every three seconds (and no fault message occurs). **(d)** we obtain the discretization $D = \{\!\{ H \}\!\}$, and **(e)** prove that the closed loop system satisfies the specification.

Theorem 6.1 (Adequate Control). *For arbitrary $r \in [0, 100]$ the closed loop system is behaviorally congruent to the specification (which does not contain the action fail). Let $\sigma = \{{}^{r,\perp}/_{x,p}\}$ and $\vec{n} = up, down$. Formally we state: $\sigma(\boldsymbol{\nu}\, \vec{n}) \leftthreetimes (H\,|\,C) \approx^{\mathrm{c}} \{{}^{r}/_{x}\}\, S$.*

Proof *(Sketch).* $\sigma(\boldsymbol{\nu}\, \vec{n}) \leftthreetimes (H\,|\,C) \approx^{\mathrm{c}} \sigma(\boldsymbol{\nu}\, \vec{n})\, \{\!\{ H \mid C \}\!\} \approx^{\mathrm{c}} \sigma(\boldsymbol{\nu}\, \vec{n})\ (D\,|\,C) \approx^{\mathrm{c}} \{{}^{r}/_{x}\}\, S$. The 1st step holds by Corollary 5.5. The 2nd step holds by applying our syntactic discretization properly. The 3rd step needs to be shown by standard analysis of simple timed systems. Since inter-process synchronization is absent the contained delay is irrelevant. □

7 Related Work and Future Work

Related Work: There are various formalisms for hybrid systems featuring different aspects. Semantical formalisms as hybrid automata [14,17] and adhoc treatment as in [19,11,10] are broadly accepted, yet non-operational for the modeling of the controller (i.e., the control strategy is not contained in the model as an algorithm). Some of HCCS's most important characteristics are (a) *the algebraic analysis* (to formally proof correspondence between specification and implementation as in Theorem 6.1): the calculi HyPA [8], χ [3], ϕ [23], $\mathrm{ACP}^{\mathrm{srt}}_{\mathrm{hs}}$ [5], HYPE, and BHPC [6] have comparable capabilities, however, some of them have a huge number of operators making the development of meta-theoretical results tough. (b) *the explicit algebraic handling of DEs* (to enable the explicit transformation

H: Hybrid System

$$H_0 \triangleq (\boldsymbol{\mu} X)\,([x < 50]\,.\,\overline{up}\langle \mathsf{L}\rangle\,.\,H_1) \mid ([x \geq 50]\,.\,\overline{up}\langle \mathsf{H}\rangle\,.\,H_1)$$

$$H_1 \triangleq down(s)\,.\,H_2$$

$$H_2 \triangleq \begin{cases} \;\; [s = +]\,.\,[t = 0 \wedge x = x \mid \dot{t} = 1 \wedge \dot{x} = 10 \mid t = 1 \mid \mathbf{0}]\,.\,H_3 \\ \mid [s = -]\,.\,[t = 0 \wedge x = x \mid \dot{t} = 1 \wedge \dot{x} = -10 \mid t = 1 \mid \mathbf{0}]\,.\,H_3 \end{cases}$$

$$H_3 \triangleq ([x \in [0, 100]]\,.\,X) \mid ([x \notin [0, 100]]\,.\,\overline{fail})$$

C: Controller

$$C_0 \triangleq (\boldsymbol{\mu} X)\, up(c)\,.\,C_1$$

$$C_1 \triangleq \overline{ob}\langle c\rangle\,.\,C_2$$

$$C_2 \triangleq \begin{cases} \;\; [p = \mathsf{H} \vee (p = \bot \wedge c = \mathsf{H})]\,.\,\overline{down}\langle -\rangle\,.\,C_3 \\ \mid [p = \mathsf{L} \vee (p = \bot \wedge c = \mathsf{L})]\,.\,\overline{down}\langle +\rangle\,.\,C_3 \end{cases}$$

$$C_3 \triangleq [p \lhd c]\,.\,X$$

S: Specification

$$S \triangleq ([x < 50]\,.\,[n \lhd \lceil (50 - x)/10 \rceil]\,.\,S_{\mathsf{L}}) \mid ([x \geq 50]\,.\,[n \lhd 1 + \lfloor (x - 50)/10 \rfloor]\,.\,S_{\mathsf{H}})$$

$$S_v \triangleq \overline{ob}\langle v\rangle\,.\,[1]\,.\,[n \lhd n - 1]\,.\,([n = 0]\,.\,[n, v \lhd 3, \mathsf{invertLH}(v)]\,.\,S_v) \mid ([n > 0]\,.\,S_v)$$

D: Discretization of H (after garbage collection)

$$D_0 \triangleq (\boldsymbol{\mu} X)\,([x < 50]\,.\,\overline{up}\langle \mathsf{L}\rangle\,.\,D_1) \mid ([x \geq 50]\,.\,\overline{up}\langle \mathsf{H}\rangle\,.\,D_1)$$

$$D_1 \triangleq down(s)\,.\,D_2$$

$$D_2 \triangleq ([s = +]\,.\,[1]\,.\,[x \lhd x + 10]\,.\,D_3) \mid ([s = -]\,.\,[1]\,.\,[x \lhd x - 10]\,.\,D_3)$$

$$D_3 \triangleq ([x \in [0, 100]]\,.\,X) \mid ([x \notin [0, 100]]\,.\,\overline{fail})$$

Fig. 3. A tank with its controller, specification and discretization

required for the discretization procedure): this is, to the best of our knowledge, only straighforwardly available to a comparable degree in HyPA and $\mathrm{ACP}^{\mathrm{srt}}_{\mathrm{hs}}$. (c) *the* CCS *based communication* (which we believe to be more natural for controller-internal communication than broadcast communication as in, e.g., BHPC, HyPA, $\mathrm{ACP}^{\mathrm{srt}}_{\mathrm{hs}}$): the formalisms χ and the CIF [26] also feature CCS based communication for the same reasons as we, however, both formalisms happen to be rather large-scale modelling formalisms trying to cover a wide range of—if not all—hybrid systems, and (d) *the greedy construction of continuous steps* (to facilitate an almost finitely branching semantics simplifying the reasoning about enabled steps greatly as explained before). HCCS, differently from ϕ, does not contain mobility [18] because mobility is primarily important for distributed rather than for embedded systems. Since none of the available calculi mentioned above offered all features that are necessary for our work (mainly value passing, computation, and explicit algebraic handling of DEs) while still being reasonably small to be able to proof the relevant meta-theorems with moderate effort we have chosen to extend CCS with only two new operators to obtain a very small and suitable, yet comparably simple, calculus instead of extending some subset of, e.g., χ or the CIF. The calculus extends CCS by constructs for modeling CTs using DEs and the realization of the greediness follows along the lines of $\mathrm{CCS}^{\mathrm{rt}}$ [7]. The abortion of continuous flows is contained for a similar reason as the disrupt operator is contained in HyPA; there, the latter is used to terminate a prefix describing the current (non-terminating) continuous evolution. Some implications of zeno behavior on bisimulation are also discussed in [9,8]. Decidability for various selections of features of hybrid systems is analysed in [2] where finite quotients of the hybrid systems are constructed w.r.t. LTL satisfaction equivalence. Deviating from the former approach in [25] a simplifying quotienting technique (using predicate abstraction and qualitative reasoning) is introduced. The resulting loss of information is kept sufficiently small to establish certain safety results.

In [1] another discretization of hybrid systems is presented. However, they are dealing with a modification of hybrid systems where the state variables are periodically sampled using digitization with finite precision and where the discrete step is not instantaneous. In this quite different class of hybrid systems, reachability is decidable and the sequences of states and actions are regular and the corresponding finite automata can be effectively constructed. In [21] the hybrid system is simplified by abstracting the continuous behavior replacing flows with flow inclusions and implementing the abstraction as a timed automaton whose reach set is a superset of the reach set of the original hybrid system, that is, as opposed to this paper, their discretization includes an overapproximation. In [24] a top-down refinement approach (based on the notion of trace inclusion for hybrid automata) with the focus on taking the variants of inexact implementations methodologically into account is introduced while the bottom-up techniques of this paper require the explicit modelling of disturbances and models are related by bisimulations.

Future Work: Our work lays the foundation for various extensions. (i) Discretization also plays a key role in the hierarchical control of embedded systems [22]. Initial results suggest that our discretization can be extended to deal properly with hierarchical abstractions. (ii) HCCS can be extended to deal with distributed reconfiguring multicontroller systems using mobility such as in swarm systems. (iii) While we are convinced that this calculus is adequate for the modeling of embedded systems, an in-depth comparison with hybrid automata and other formailsms (e.g., the CIF) is called for to be able to transfer the obtained results in a sound manner. Presumably hybrid automata deviate only in their handling of enabled discrete steps: where hybrid automata allow discrete steps (may) the calculus HCCS enforces these steps (must) which is similar to the ASAP semantics in [27]. (iv) For more complex DEs, an extension could include the usage of known over- and under approximations of the sets of traversed valuations and the final valuation of a CTs for the execution of meaningful discretizations. (v) Furthermore, we expect it to be straightforward to extend our discretization such that it covers aspects like uncertain input, disturbances, and less perfect hybrid systems as with inexact synchronization [4] and imperfect tests [27].

References

1. Agrawal, M., Thiagarajan, P.S.: The discrete time behavior of lazy linear hybrid automata. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 55–69. Springer, Heidelberg (2005)
2. Alur, R., Henzinger, T.A., Lafferriere, G., Pappas, G.J.: Discrete abstractions of hybrid systems. Proceedings of the IEEE 88(7), 971–984 (2000)
3. Baeten, J.C.M., van Beek, D.A., Cuijpers, P.J.L., et al.: Model-based engineering of embedded systems using the hybrid process algebra χ. ENTCS 209, 21–53 (2008)
4. Beohar, H., Cuijpers, P.J.L., Baeten, J.C.M.: Design of asynchronous supervisors. CoRR, abs/0910.0868 (2009)

5. Bergstra, J.A., Middelburg, C.A.: Process algebra for hybrid systems. Theor. Comput. Sci. 335(2-3), 215–280 (2005)
6. Brinksma, E., Krilavičius, T.: Behavioural hybrid process calculus. TR-CTIT-05.45, CTIT, University of Twente (2005)
7. Cleaveland, R., Lüttgen, G., Natarajan, V.: Handbook of Process Algebra. Ch.12. Elsevier Science Inc., Amsterdam (2001)
8. Cuijpers, P.J.L.: Hybrid Process Algebra. PhD thesis (2004)
9. Cuijpers, P.J.L., Reniers, M.A.: Topological (bi-)simulation. ENTCS 100, 49–64 (2004)
10. Cuijpers, P.J.L., Reniers, M.A., Heemels, W.P.M.H.: Hybrid transition systems. TR, Department of Computer Science, TU/e (2002)
11. Davoren, J.M., Moor, T., Nerode, A.: Hybrid control loops, a/d maps, and dynamic specifications. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 149–163. Springer, Heidelberg (2002)
12. Fränzle, M., Herde, C.: Hysat: An efficient proof engine for bounded model checking of hybrid systems. FMSD 30(3), 179–198 (2007)
13. Galpin, V., Bortolussi, L., Hillston, J.: Hype: A process algebra for compositional flows and emergent behaviour. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 305–320. Springer, Heidelberg (2009)
14. Henzinger, T.A.: Verification of digital and hybrid systems. NATO ASI Series F 170, 265–292 (2000)
15. Henzinger, T.A., Ho, P., Wong-Toi, H.: Hytech: A model checker for hybrid systems. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 460–463. Springer, Heidelberg (1997)
16. Henzinger, T.A., Horowitz, B., Majumdar, R., Wong-Toi, H.: Beyond hytech: Hybrid systems analysis using interval numerical methods. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 130–144. Springer, Heidelberg (2000)
17. Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid I/O automata. Inf. Comput. 185(1), 105–157 (2003)
18. Milner, R.: Communication and concurrency. Prentice-Hall, Inc., Englewood Cliffs (1989)
19. Moor, T., Raisch, J., O'Young, S.: Discrete supervisory control of hybrid systems based on l-complete approximations. DEDS 12(1), 83–107 (2002)
20. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reasoning 41(2), 143–189 (2008)
21. Puri, A., Varaiya, P.: Verification of hybrid systems using abstractions. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S.S. (eds.) HS 1994. LNCS, vol. 999, pp. 359–369. Springer, Heidelberg (1995)
22. Raisch, J., Moor, T.: Hierarchical Hybrid Control of a Multiproduct Batch Plant. LNCIS, vol. 322, pp. 99–216. Springer, Heidelberg (2005)
23. Rounds, W.C.: A spatial logic for the hybrid π-calculus. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 508–522. Springer, Heidelberg (2004)
24. Stauner, T.: Discrete-time refinement of hybrid automata. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 407–420. Springer, Heidelberg (2002)
25. Tiwari, A.: Abstractions for hybrid systems. FMSD 32(1), 57–83 (2008)
26. van Beek, D.A., Reniers, M.A., Schiffelers, R.R.H., et al.: Foundations of a compositional interchange format for hybrid systems. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 587–600. Springer, Heidelberg (2007)
27. De Wulf, M., Doyen, L., Raskin, J.: Almost asap semantics: from timed models to timed implementations. Formal Asp. Comput. 17(3), 319–341 (2005)

Model-Based Dependability Analysis of Programmable Drug Infusion Pumps*

Sriram Sankaranarayanan, Hadjar Homaei, and Clayton Lewis

University of Colorado, Boulder, CO
first.lastname@colorado.edu

Abstract. Infusion pumps are commonly used in home/hospital care to inject drugs into a patient at programmable rates over time. However, in practice, a combination of faults including software errors, mechanical failures and human error can lead to catastrophic situations, causing death or serious harm to the patient. Dependability analysis techniques such as failure mode effect analysis (FMEA) can be used to predict the worst case outcomes of such faults and facilitate the development of remedies against them.

In this paper, we present the use of model-checking to automate the dependability analysis of programmable, real-time medical devices. Our approach uses timed and hybrid automata to model the real-time operation of the medical device and its interactions with the care giver and the patient. Common failure modes arising from device failures and human error are modeled in our framework. Specifically, we use "mistake models" derived from human factor studies to model the effects of mistakes committed by the operator. We present a case-study involving an infusion pump used to manage pain through the infusion of analgesic drugs. The dynamics of analgesic drugs are modeled by empirically validated pharmacokinetic models. Using model checking, our technique can systematically explore numerous combinations of failures and characterize the worse case effects of these failures.

1 Introduction

The delivery of critical medications through "smart" infusion pumps has become quite commonplace in home/hospital medical care. Yet, there have been numerous fatal or near-fatal incidents due to errors such as software error, pump malfunctions and human operator errors [1, 20, 24, 44, 46]. In this paper, we present a framework using formal verification techniques over timed and hybrid automata models to analyze the worst case effects of combinations of human and machine errors on the safety of the patient. Our framework systematically considers combinations of failure modes and provides quantitative predictions of the worst case outcomes for each combination. The failure modes considered here include mechanical failures in the pump, air bubbles, occlusions, empty vials, data entry errors and a failure to respond to alarms in a timely manner.

Our approach models the infusion pump by composing a simple hybrid automaton, incorporating a model of the pump, along the lines of the generic infusion pump model

* This material is based upon work supported by the National Science Foundation (NSF) under award no. 1035845.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 317–334, 2011.

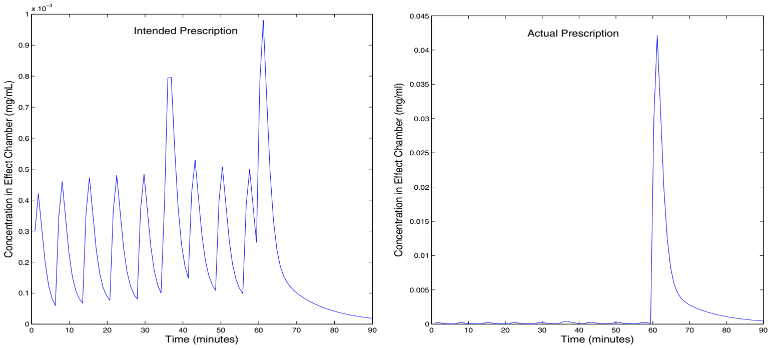


Fig. 1. Effect chamber concentration in the patient predicted by a pharmacokinetic model for (**left**) originally intended infusion (y-axis range: $[0, 1 \times 10^{-3}]$ mg/ml), and (**right**) actual infusion as a result of user error (y-axis range: $[0, 0.045]$ mg/ml)

proposed by Arney et al. [2] and the patient using a pharmacokinetic model that describes the concentration of the drug in various parts of the patient's body as the infusion proceeds over time [26, 39, 41]. Mechanical failures are modeled using failure mode variables and transitions. Our approach incorporates models of pump programming errors based on generic human error models commonly studied in the human factors community [22, 27, 36] along with the results of studies and reports on the common types of errors committed by caregivers while programming infusion pumps [3, 4, 28, 37, 47]. The composition of these models yields an affine hybrid automaton. The composed model is analyzed using bounded model checking (BMC) to compare the concentration of the drug in the original failure-free (original) model with the concentration in the failure-enabled model [5]. Specifically, the use of BMC allows us to search for executions, wherein, the timing of the faults and external disturbances yield the maximum (or minimum) possible concentration of the drug being infused in the patient's body. Comparing the range of drug concentrations resulting from the original fault-free model with the range obtained from the fault-enabled models provides a quantitative measure of the potential effects of the fault on the safety of the patient. Thus, the output of our analysis can be used as the starting point to help risk analysts construct dependability models in the form of a fault trees or failure mode effects analysis (FMEA) tables [17].

1.1 Motivation

We consider a brief summary of a fatal overdose incident due to wrong programming of an infusion pump, revealing some of the key hazards to patient safety due to programmable infusion pumps [20].

Event: The intended prescription ordered a 50 mcg/ml dose of the opioid (pain-killer) Fentanyl through a patient controlled analgesic (PCA) pump. A PCA pump allows the patient to request a preset dose of the drug by pressing a key. PCA pumps also enforce a set minimum lockout times between two requests. In this case, PCA request dose was 10 mcg with 6 minute lockout between doses. An extra bolus dose of 20 mcg (infused at maximum possible rate) was prescribed in case the pain level was high.

The device required as input: (a) the concentration of the drug stated in the vial, which was incorrectly entered as 1 mcg/ml instead of the correct concentration of 50 mcg/ml[1].This error may have resulted, in part, due to a "feature" in the device that silently reverted back to a previously entered value if an entry was not confirmed within some time limit; and (b) The demand dose for PCA was incorrectly set to 0.1 mcg instead of 10 mcg.

Outcome: Upon each PCA request the pump infused 0.1 mcg at 1 mcg/ml = 0.1 ml by volume. However, the actual drug concentration in the vial was 50 mcg/ml. This meant that the patient received 50 mcg/ml × .1 ml = 5 mcg upon each PCA demand dose. Fortunately, this was half the originally intended amount: the error in concentration had nullified that in the PCA demand dose. However, the pain persisted and a bolus dosage of 20 mcg was entered without correcting the error in the drug concentration. The pump infused 20 mcg at 1 mcg/ml = 20 ml of the drug into the patient. However, the patient actually received a dose of 20 ml × 50 mcg/ml = 1000 mcg of the drug, which resulted in a 50 × overdose. This overdose proved ultimately fatal.

Approach: Our approach can predict the scenario outlined as follows:

1. We consider the effect of numerous common pump programming errors on the data entered, as described in Section 4.
2. For each such error, our approach compares the intended vs. actual concentrations of the drug in the patient's body. To achieve this, we use hybrid automata models for the pump (Section 3.1) and pharmacokinetic models (Section 3.2).
3. Finally, we use bounded-model checking (BMC) using SMT solvers on a fixed time step approximation of the composed hybrid model to compare the possible range of drug concentrations in the original fault-free execution and the modified fault-enabled execution.

Figure 1 shows the concentration of the drug in the effect compartment of a pharmacokinetic model as a function of time both for the originally intended infusion and the infusion that occurs as a result of the error. The kinetics used for Fentanyl are as reported in the literature (Cf. Vuyk et al. [48]). Based on a comparison of the original with the faulty trace, it is seen that the new prescription results in a overdose of roughly 40 times the original concentration.

While the effects of an extreme scenario described above are easy to predict *qualitatively*, our framework provides three key advantages: It systematically explores a large range of possible operator mistakes and hardware/software failures. Secondly, our approach can explore the space of timings of the faults and the patient actions such as requesting a PCA bolus that can cause a worst-case outcome. Finally, our approach can quantitatively predict the worst-case outcome of a combination of faults on the patient using empirically validated pharmacokinetic models to predict the drug concentrations.

2 Preliminaries

We briefly recall the hybrid automaton model [21] and describe the use of formal verification to model faults and analyze dependability properties of dynamic systems [7].

[1] Note that recent models use barcode readers to obtain this information from the vial.

Hybrid automata are commonly used to model the composition of a discrete (software-based) controller interacting with a continuous environment [21].

Definition 1. *A hybrid automaton $\mathcal{H}$ is a tuple $\langle \boldsymbol{x}, \boldsymbol{d}, \mathcal{L}, \mathcal{T}, \mathcal{F}, \mathcal{I}, \Theta \rangle$, wherein,*

- $\boldsymbol{x} \in \mathbb{R}^n$ *refers to a vector of* continuous system variables.
- $\boldsymbol{d} \in \mathbb{R}^m$ *refers to a set of continuous external input (disturbance) variables.*
- $\mathcal{L}$ *refers to a finite set of* discrete locations *or modes.*
- $\mathcal{T}$ *refers to a set of discrete* transitions *(or mode changes). Each transition $\tau \in \mathcal{T}$ is of the form $\langle \ell, m, \rho_\tau[\boldsymbol{x}, \boldsymbol{d}, \boldsymbol{x}'] \rangle$, wherein, $\ell, m \in \mathcal{L}$ refer to the pre and post locations respectively, and ρ_τ is an assertion, representing the transition relation relation between the current state ($\boldsymbol{x}$) and the next state variables ($\boldsymbol{x}'$).*
- $\mathcal{D}$ *associates each location $\ell \in \mathcal{L}$ with a system of ordinary differential equations (ODEs) $\frac{d\boldsymbol{x}}{dt} = F_\ell(\boldsymbol{x}, \boldsymbol{d})$, wherein $F_\ell : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ is a Lipschitz continuous function over $\boldsymbol{x}, \boldsymbol{d}$.*
- $\mathcal{I}$ *maps each location to a domain $\mathcal{I}(\ell) \subseteq \mathbb{R}^n$.*
- $\Theta \subseteq \mathcal{L} \times \mathbb{R}^n$ *refers to the initial conditions.*

The semantics and properties of hybrid automata are discussed elsewhere [21, 43].

2.1 Dependability Analysis

Dependability analysis of a safety critical systems identifies potential faults that may occur in the system and the worst-case effects of these faults on the safety, reliability and the performance of the system as a whole [17]. It has played a significant role in the design of safety critical control systems including avionics, satellites and nuclear reactors. Traditionally, dependability analysis techniques are *static* in nature, wherein the timing of the faults and the evolution of system state are not modeled.

Fault Tree: A fault tree is an enhanced circuit with logic gates (AND/OR/XOR) and other gates for modeling fault propagation, that computes the possibility of a particular type of system level failure as a function of the individual failure modes. Fault trees enable the computation of failure probabilities and expected time to failure as a function of the probability of the individual faults.

Failure Mode Effects Analysis: A *failure mode effects analysis* (FMEA) table lists different failure modes, mapping each failure mode to its causes, detection mechanisms, severity, expected frequency of occurrence and possible mitigations. FMEA is performed by risk analysts during system or process design.

In many complex systems, however, the timing of the events and the dynamics of the system are key to understanding the effects of failures. As a result, there has been much work on dynamic reliability techniques using a combination of continuous, discrete and stochastic models (including hybrid automata models [35]) in conjunction with techniques such as Monte-Carlo simulation for predicting failure probabilities [42].

Formal Dependability Analysis: There has been much recent progress in the use of formal verification techniques such as model checking for automating dynamic dependability analyses [7, 8]. We provide a brief description of these approaches to model and reason about the effect of faults.

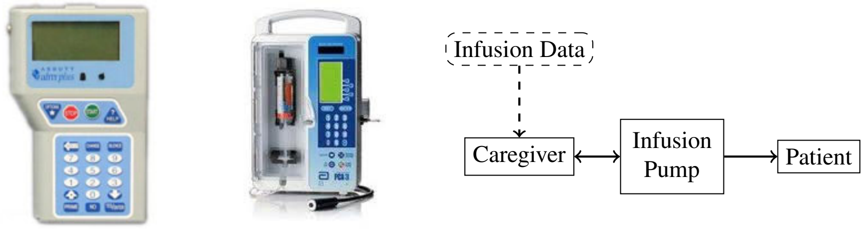


Fig. 2. (left) Examples of commercial infusion pumps and **(right)** major components of the drug infusion model

Let $\mathcal{H}$ represent the model of the original system over state variables $\boldsymbol{x}$. The effect of failure is modeled by a combination of non-deterministic Boolean valued *failure mode variables* $f_1, \ldots, f_m$ and extra locations and transitions for modeling the system's operation upon failures to obtain a system $\mathcal{H}_f$ that can model the effects of failures. The primary purpose of the failure mode variables is to guard the transition to a subsystem modeling failure. Let $\varphi_1, \ldots, \varphi_k$ represent assertions that signify safety properties of the system $\mathcal{H}$. We assume that the original system satisfies these properties.

Symbolic model checking, or any reachability analysis technique, can be used on the augmented model $\mathcal{H}_f$ to search for reachable states that violate some of the safety properties [12]. Let $s \in [\![\varphi_j]\!]$ be a reachable error state in $\mathcal{H}_f$. Since $\mathcal{H}$ itself is assumed safe, we note that some subset F_s of the failure mode variables were enabled to reach the error state s. The overall analysis systematically collects such subsets F_s, which are termed cut-sets. Bozzano et al. also present techniques for finding minimal cut-sets of failure mode variables. The collection of failure modes that can lead a system to violate a property can be presented in the form of a fault tree or a FMEA table to help the panel of risk analysts better understand all the possible threats. This analysis has been integrated in the tool SLIM [8]. Our work is directly inspired by these efforts.

However, in our work, the modeling of human operator error is a key aspect of medical device failures. Mistakes such as data entry errors, unit conversion mistakes and misinterpretation of prescription labels modify the parameters of our model. Secondly, while it is possible to specify safety properties in terms of limits on how much drug can be infused, it is more natural in our setting to compare the range of doses that can be achieved using the original parameters (the intended infusion) vs. the range achieved by the fault-enabled parameters. Finally, we investigate pharmacokinetic models with dynamics that can be affine or non-linear.

3 System Models

Drug infusion pumps are commonly used in home/hospital medical care to inject drugs directly into the blood stream of a patient. Most infusion pumps are "programmable" by the caregiver, whose role consists of entering vital information from the prescription to the start of the infusion through a user interface. Infusion pumps support various *delivery modes*, as shown in Table 1. Combinations of these modes (eg., Bolus + PCA+ Cont. or Bolus + Cont.) are supported by many infusion pump models. The interface

Table 1. Basic infusion modes supported by pump models along with the parameters relevant to each mode

Mode	Description	Parameters
Continuous	Infusion at a continuous rate	rate cRate (mcg/min) concentration c (mcg/ml)
Bolus	Fast infusion of drug volume.	dosage w_b (mcg) concentration c (mcg/ml)
Patient-Control (PCA)	A bolus administered upon patient request	amtPerRequest (mcg) lockout (min) dose limit (mcg/hr) concentration (mcg/ml)

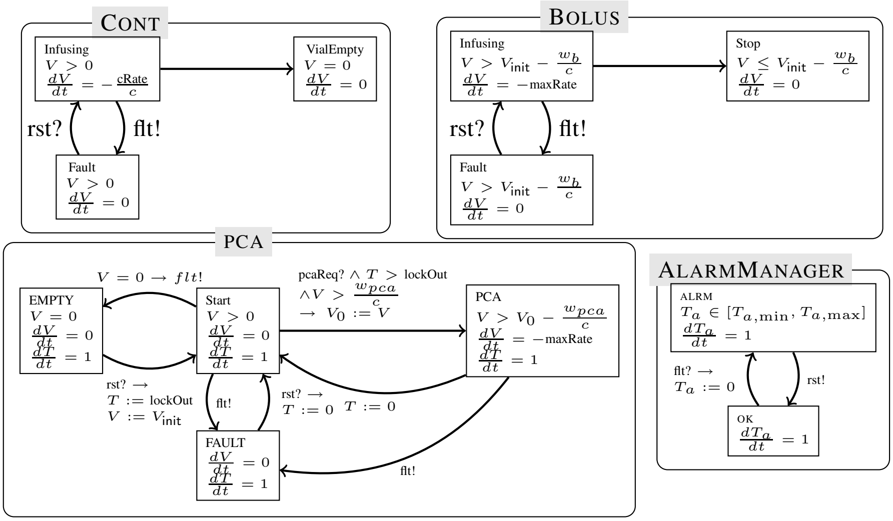
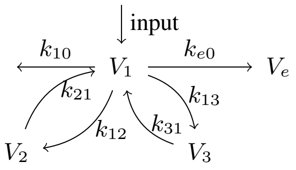

CONT
Infusing $V > 0$ $\frac{dV}{dt} = -\frac{\text{cRate}}{c}$
VialEmpty $V = 0$ $\frac{dV}{dt} = 0$
rst?
flt!
Fault $V > 0$ $\frac{dV}{dt} = 0$
BOLUS
Infusing $V > V_{\text{init}} - \frac{w_b}{c}$ $\frac{dV}{dt} = -\text{maxRate}$
Stop $V \leq V_{\text{init}} - \frac{w_b}{c}$ $\frac{dV}{dt} = 0$
rst?
flt!
Fault $V > V_{\text{init}} - \frac{w_b}{c}$ $\frac{dV}{dt} = 0$
PCA
$V = 0 \rightarrow flt!$
EMPTY $V = 0$ $\frac{dV}{dt} = 0$ $\frac{dT}{dt} = 1$
Start $V > 0$ $\frac{dV}{dt} = 0$ $\frac{dT}{dt} = 1$
pcaReq? $\wedge$ $T >$ lockOut $\wedge V > \frac{w_{pca}}{c}$ $\rightarrow V_0 := V$
PCA $V > V_0 - \frac{w_{pca}}{c}$ $\frac{dV}{dt} = -\text{maxRate}$ $\frac{dT}{dt} = 1$
rst? $\rightarrow$ $T :=$ lockOut $V := V_{\text{init}}$
flt!
rst? $\rightarrow$ $T := 0$
$T := 0$
FAULT $\frac{dV}{dt} = 0$ $\frac{dT}{dt} = 1$
flt!
ALARMMANAGER
ALRM $T_a \in [T_{a,\min}, T_{a,\max}]$ $\frac{dT_a}{dt} = 1$
flt? $\rightarrow$ $T_a := 0$
rst!
OK $\frac{dT_a}{dt} = 1$

Fig. 3. Pump model for continuous, bolus, PCA infusion modes and the alarm manager component

to the infusion pump includes protections from tampering such as a physical key to prevent reprogramming by the patient once the infusion has commenced.

The major components that are universally present in most pump models are: (1) A user-interface that allows the caregiver to "program" the infusion by entering the infusion mode and the data pertaining to each mode. (2) A syringe or bag loaded with a given total volume of the infused drug at some known concentration. (3) Various alarms that are sounded upon conditions such as airbubbles, blockages, pump failures, empty vials and so on. It is the responsibility of caregiver to reset the system to resume the infusion upon an alarm. (4) In Patient Controlled Analgesic (PCA) pumps, the patient can request a bolus of the drug by pressing a key.

Figure 2 shows some examples of infusion pumps and the three major components that we model as part of the dependency analysis of a drug infusion pump. We first discuss how various components are modeled in our approach and then present failure



Var	Remarks
V_i	Volume of the i^{th} chamber (ml).
$x_{1,2,3}$	Concentration in the i^{th} chamber (mcg/ml).
x_e	Concentration in the effect chamber (mcg/ml).
u	instantaneous infused concentration (mcg/min).
k_{ij}	kinetic diffusion param. from V_i and V_j.
k_{10}	param. for drug leaving chamber V_1.

Fig. 4. Three compartment pharmacokinetic model with an effect compartment. The kinetic parameters associated with each edge is shown.

modes associated with the caregiver and the infusion pump itself. We refer the reader to the work of Arney et al. as part of the generic infusion pump modeling project for a classification of existing models into many different types and their proposed generic (and comprehensive) infusion pump model [2]. In this work, we use a simplified version inspired by the Arney et al. infusion pump model. Our model is augmented with a patient model for the purposes of dependability analysis.

3.1 Pump Model

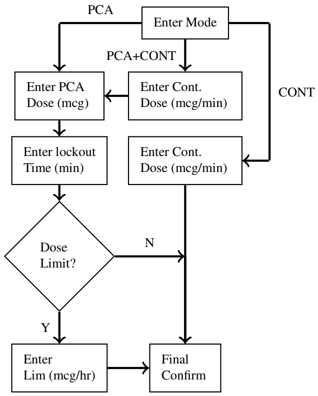
The pump itself is modeled by a hybrid automaton whose discrete modes describe the infusion mode. The continuous variables describe the rate at which the drug is leaving the vessel (and entering the patient) and the volume left in the vessel. We recall the three infusion modes as described in Table 1. The (simplified) hybrid automaton model for these modes are shown in Figure 3. We note that in each case, the pump component has a variable V representing the volume of the pump. Parameter V_{init} represents the volume in the syringe at the start of the infusion, w_b represents the ordered bolus weight and c represents the drug concentration, cRate the programmed rate of continuous infusion and maxRate the maximum possible rate at which the drug can be delivered (for a bolus/pca infusion). The infusion mode and the parameters w_b,c,w_{pca} and lockOut can be read in through barcode readers or entered through the user interface.

The pump issues internal fault events upon detection of a bubble, an occlusion or a pump failure. As a result, it transitions to a mode where the dynamics model the stoppage of the pump. These faults are transient in nature and result in an alarm being issued to the human operator. The operator then remedies the situation causing the fault and resets the operation back to the point in the infusion where the fault occurred. We assume in our model that all faults end up stopping the infusion. However, in many models, the infusion can often continue during some faults like low charge in the battery.

Alarm Manager: The alarm management model is shown in Fig. 3. We model a single alarm type that can be issued for various reasons. Our model assumes that alarms are handled within some fixed time interval by resetting the system to some fixed state. Our model uses timers to track the amount of time the current state is in a given mode. This allows us to model delays in the reset transition due to inattention to alarms.

3.2 Patient Model

The model of the patient is intended to capture the effect of the infusion in the patient's body. Since infusion pumps are able to deliver drugs at various pre-programmed



Prompt	Value
Enter Mode	PCA + Cont.
Enter Cont. Dose	0.1 mcg/min
Enter PCA Dose	1 mcg
Enter Lockout	5 min
Dose Limit?	Y
Enter Limit	10 mcg/hr

Fig. 5. (**Left**) Schematic interaction diagram for a medical infusion pump. Diagram is based roughly on the abbott PCA3 infusion pump. (**Right**) An example prescription entered through the interface.

concentrations over time, we require a dynamical model of the drug as it is absorbed (and possibly metabolized) by the various cells in the patient's body and removed from the blood stream. We therefore use pharmacokinetic models to capture such effects over time [26, 39, 41].

A pharmacokinetic model uses multiple *compartments*, representing the various systems such as the vascular system, organs such as the liver, kidneys and the skin. The number of compartments used in a model depends on the specific drug whose kinetics are being modeled. The model is described by an ordinary differential equation whose variables represent the concentration of the drug in the various compartments. The dynamics of the model are given by

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_e} \end{pmatrix} = \begin{pmatrix} -(k_{10}+k_{12}+k_{13}) & k_{21}\frac{V_2}{V_1} & k_{31}\frac{V_3}{V_1} & 0 \\ k_{12}\frac{V_1}{V_2} & -k_{21} & 0 & 0 \\ k_{13}\frac{V_1}{V_3} & 0 & -k_{31} & 0 \\ k_{e0} & 0 & 0 & -k_{e0} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_e \end{pmatrix} + \begin{pmatrix} \frac{1}{V_1}u \\ 0 \\ 0 \\ 0 \end{pmatrix} .$$

The values of the parameters may vary, depending on the drug whose dynamics is under investigation. Representative values are obtained as a result of studies conducted on a group of patients (Cf. [18, 30], for instance).

4 Human Interaction Models

In this section, we present a methodology for modeling human operator actions and the mistakes committed in the course of their interactions. In the setting of the infusion pump study, a human operator of the infusion pump is the care giver who is responsible for programming the infusion parameters at the start of the infusion and responding to alarms raised by faults such as air bubbles and occlusions in a timely manner.

Table 2. Major pump programming errors found in the literature

	Error Description	References
1	Fields in the prescription are misinterpreted	[37, 40, 47]
2	Dosage Data Entry Error	[20, 37]
3	Unit Conversion Calculation Errors	[1, 4, 24, 28]
4	Pump mode selection errors	[3]

User Interface: Infusion pumps communicate with human operators by means of an user interface that displays messages to prompt the human operator and allows the user to enter values. Due to a lack of standardization of such interfaces, different devices offering the same set of basic functionalities may present a variety of interfaces for the same task [2]. Figure 5 shows an example interaction flowchart for an infusion pump based on an existing commercial model.

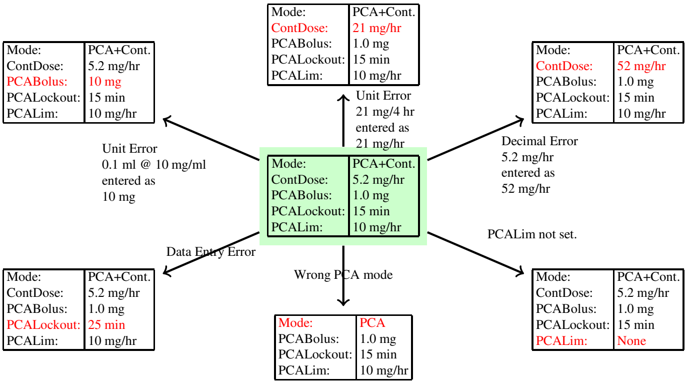
Prescriptions: Infusions of analgesics such as Fentanyl or morphine are tightly regulated, often requiring a prescription that carefully specifies the allowed dosage and the recommended mode of infusion. The details of a how a given infusion is prescribed can vary significantly, in general [4].

Example 1. An example prescription may call for a basal continuous dosage of 6 mcg/hr (=.1 mcg/min) of 50 mcg/ml of morphine and a PCA delivery of 1 mcg per request with a lockout interval of 5 minutes with an overall limit of 10 mcg each hour. It is the role of a care giver (often a registered nurse) to program this prescription using the infusion pump interface as shown in Figure 5. The data entered by the user is also shown.

4.1 Modeling Operator Mistakes

We discuss common mistakes that are committed by the operator during the course of an interaction and present a generic modeling system for these interaction errors. In theory, any *unintentional* deviation from a correct sequence of actions needed to deliver a given prescription of a drug to a patient can be classified as a mistake. There have been extensive studies by human factors experts and psychologists on the cognitive factors underlying human error [22, 23, 36]. Furthermore, the specific types of mistakes that operators commit while operating an infusion pump have also been well-studied and documented [3, 9, 20, 24, 28, 37, 40, 47]. Table 2 lists some of the common pump programming errors as reported in the literature. A comprehensive list of many types of hazards (including human error) for a generic infusion pump model has been put together by Arney et al. [3].

Data Entry Errors: In general, data entry errors depend on the type of interface used by the pump. For instance, empirical studies have shown that for pumps with keypads, the most harmful keypad entry errors include decimal point errors. Therein, a decimal point may either be inserted or deleted from the number entered due to the proximity of the decimal point key or variations in its placement across different interfaces [44]. Quirks in the interface such as reverting to a default value or a previously entered prescription also contribute to data entry errors (Cf. Section 1.1).



Mode:	PCA+Cont.
ContDose:	5.2 mg/hr
PCABolus:	1.0 mg
PCALockout:	15 min
PCALim:	10 mg/hr

Mode:	PCA+Cont.
ContDose:	21 mg/hr
PCABolus:	1.0 mg
PCALockout:	15 min
PCALim:	10 mg/hr

Mode:	PCA+Cont.
ContDose:	5.2 mg/hr
PCABolus:	10 mg
PCALockout:	15 min
PCALim:	10 mg/hr

Mode:	PCA+Cont.
ContDose:	52 mg/hr
PCABolus:	1.0 mg
PCALockout:	15 min
PCALim:	10 mg/hr

Mode:	PCA+Cont.
ContDose:	5.2 mg/hr
PCABolus:	1.0 mg
PCALockout:	25 min
PCALim:	10 mg/hr

Mode:	PCA
PCABolus:	1.0 mg
PCALockout:	15 min
PCALim:	10 mg/hr

Mode:	PCA+Cont.
ContDose:	5.2 mg/hr
PCABolus:	1.0 mg
PCALockout:	15 min
PCALim:	None

Fig. 6. Transformation of a prescription (shown center) through various mistakes

Unit Conversion Errors: Prescriptions may be provided in units that differ from those required by the interface, requiring a calculation to be carried out for unit conversion. As a common example, an interface may require inputs in `mg/hr` while the prescription orders a dosage in `mcg / kg/min`. The conversion from the latter requires multiplying by the patient weight in `kg` and multiplying by 60 to convert from `min` to `hr` and finally dividing by 1000 to convert from `mcg` to `mg` [1, 4]. Some "calculator pumps" enable caregivers to carry out this calculation using the interface. Other types of pumps can accept inputs in one of many different choice of units. Yet, these models are not without hazards. A common error occurs when the difference between the originally prescribed units and the entered units is simply ignored, or the wrong units are ascribed to a dose. For example, a prescription that calls for `X mg` over `Y days` can be entered as `X mg/hr`, leading to a 24 fold overdose [24].

This leads to three types of errors: (a) the wrong calculation may be carried out (eg., divide by 60 instead of multiplying), (b) the right calculation may be carried out to yield a wrong result nevertheless or (c) the difference in units may be ignored and the original value entered as is. Unit conversion errors have been well documented especially in the context of infusion pump programming errors. Bates et al. presents a detailed study of the variability in the units used to prescribe dosages [4]. Furthermore, Lesar reports on a study of such unit conversion calculations to characterize the types of errors, their rates and effects [28].

Mode Selection Errors: It is often possible that pump can be misprogrammed by choosing the wrong delivery mode in the first place. A common example in PCA pumps is the choice of a "PCA only" mode where a "PCA+Cont." mode was ordered in the first place. Often, a wrong mode selection error can be detected during data entry if the pump prompts the user for data that is not part of the original prescription. The presence of too much redundant data in the prescription label is often a factor. This phenomenon is discussed in the context of infusion pumps by Thimbleby [46].

4.2 Mistake Models

We now describe a simple framework to model the effect of operator mistakes that will be used later in our overall analysis framework to predict worst case outcomes of human operator error. We may view the effects of a mistake simply as a *transformation* between the *intended* prescription and the *actual* prescription that results due to the mistake being committed [19, 22, 36]. Therefore, we simulate pump programming mistakes through set of transformations, wherein each transformation takes the original prescription as input and yields a set of possible "mistaken prescriptions" that could result, as if the mistake were actually committed. We illustrate this through an example.

Example 2. Figure 6 shows various transformations of an original prescription due to the errors such as mode selection error, data entry errors, calculation errors and other pump programming errors. For convenience, we have shown transformations due to single mistakes. However, it is possible to consider multiple mistakes as part of our mistake model by composing transformations.

5 Analysis Framework

We will now describe the framework used to explore possible worst case scenarios that may occur due to the presence of faults due to mechanical failures and human error.

The inputs to the analysis include the model of the pump, the patient and the data for the original prescription. The overall analysis has two parts: (a) Identify all failure modes. Machine/human errors in the model are identified by analyzing the process of prescribing the drug and the interface used to enter the data into the infusion pump; and (b) For each combination of k or less failure modes, we use model checking to search for runs of the model that exhibit the maximum values of the drug concentrations. Specifically, the model checker searches in the space of possible failure times for the various enabled failure modes and the timing of the PCA requests by the patient.

Analysis of Failure Mode Combinations: The overall idea behind our analysis framework is to compare the range of drug concentrations possible for the *reference model*, assuming the absence of faults and a *fault-enabled* model, wherein some pre-defined combination of faults are enabled. The focus of the comparison in our infusion pump case-study is the variable x_e in the patient model that represents the concentration of the drug in the effect compartment.

Fixed Time Step Approximation: Our analysis uses bounded model checking (BMC) [5] using SMT solvers [13, 31] on a fixed time step approximation of the model to find worst-case scenarios that can potentially maximize or minimize the value of x_4 at some time instant. We now briefly define the notion of a Δ time step approximation. We use some basic results for affine ODEs to discretize fixed time step approximations.

Definition 2 (Fixed Time Step Runs). *Let $\mathcal{H}$ be an affine hybrid automaton. A run σ of $\mathcal{H}$ is said to conform to a fixed time step of Δ iff all discrete transitions in σ are taken at time instants that are integral multiples of Δ.*

```
Input:
    Original Prescription Data.
    Set of enabled failure modes.
Output:
    Worst-case execution traces

Compute range of effect chamber concentrations over original prescription: $[x_{\min}, x_{\max}]$.
Simulate pump programming errors to create faulty prescription.
Create BMC encoding $\Psi_M$ using faulty prescription parameters
Compute range $[y_{\min}, y_{\max}]$ for effect chamber concentration (using an SMT solver).
Compare $[x_{\min}, x_{\max}]$ with $[y_{\min}, y_{\max}]$ to compute worst-case deviations.
```

Fig. 7. Overview of Analysis Algorithm

A Δ time step approximation of a hybrid automaton for some time step $\Delta > 0$ considers only those runs σ that conform to a fixed time step of Δ.

It is easy to see that a fixed time step leads to an under approximation of the original event-triggered semantics. I.e, any (finite or infinite) run under the fixed time step approximation is also a run of the original system. On the other hand, there may be transitions that cannot be taken at integral multiples of Δ. Therefore, the choice of Δ needs to be small enough to retain most of the behaviors of the original system.

Assuming that the dynamics are of the form $\frac{d\boldsymbol{x}}{dt} = A\boldsymbol{x} + \boldsymbol{u}$, where A is invertible and the value of $\boldsymbol{u}$ remains constant during any time step interval, we conclude that

$$\boldsymbol{x}(t + \Delta) = M\boldsymbol{x}(t) + N\boldsymbol{u}, \text{ wherein } M = e^{\Delta A}, \text{ and } N = A^{-1}(e^{\Delta A} - I)\,.$$

Note that the pharmacokinetic models considered here satisfy the conditions required for the discretization.

Bounded Model Checking: Using a fixed time step Δ allows us to under approximate the original model in terms of a purely discrete model. Such a model is amenable to many verification techniques for discrete transition systems without the need to deal with the effects of ODEs, directly. For our study, we chose to use Bounded Model Checking (BMC) [5]. Using BMC, we encode the set of all runs of the model up to some fixed depth limit k by means of a logical formula in linear arithmetic. This formula is then solved by powerful SMT solvers such as Yices and Z3 [13, 16, 31].

6 Experiments

We report on an experimental evaluation of the ideas in this paper. The experimental evaluation focuses on the study of the infusion of the opioid Remifentanil through an infusion pump interface based roughly on existing commercial model — the Abbott PCA III infusion pump. Figure 8 shows the data fields entered through the pump interface and some of the calculations performed by the pump based on the entered data to derive model parameters.

Data	Range	Value Used	Entry Method
DrugConc	50 mcg/ml	50	Barcode
Weight	[30-120] kg	60	Caregiver
PCAMode	{PCA-CONT, PCA-ONLY }	PCA-CONT	Caregiver
ContDose	[0 - 0.5] mcg/kg/min	0.1	Caregiver
PCABolus	[0.01 - 1] mcg/kg	0.5	Caregiver
Lockout	[5-20] min	10	Caregiver
DoseLimit	[3-10] req/hr or no lim	4	Caregiver

Field	Equation
CRate(ml/min)	$\begin{cases} \frac{\text{ContDose}*\text{Weight}}{\text{DrugConc}} & \text{if PCAMode} = \text{PCA-CONT} \\ 0 & \text{if PCAMode} = \text{PCA-ONLY} \end{cases}$
PCABolusVol(ml)	$\frac{\text{PCABolus}*\text{Weight}}{\text{DrugConc}}$
Lockout	As input
DoseLim	As input

Fig. 8. Data fields entered in a PCA infusion pump along with limits on the data values used in our simulations and (**bottom**) calculations performed to derive infusion pump parameters from entered data

Mistake	Effect
Mode selection error (PCA-CONT vs. PCA-ONLY)	CRate' = 0
Weight entry error (weight in lb entered as kg)	CRate' = 2.2 * CRate PCABolusVol' = 2.2 * PCABolusVol
Weight entry error (weight kg entered as lb)	CRate' = CRate/2.2 PCABolusVol' = PCABolusVol/2.2
Unit Error (mcg/kg/hr entered as mcg/kg/min)	CRate' = 60 * CRate

Fig. 9. Possible data entry mistakes

The pump and the human patient models are as discussed in Section 3. We consider a set of human mistakes based on the interface used for data-entry as shown in Figure 9.

Implementation: We have implemented our analysis on top of a hierarchical, guarded command modeling language similar to existing tools such as SLIM that allows us to model the various components, faults and their interactions [8]. We compile and discretize our model and generate constraints to represent bounded depth executions. These constraints were solved using the solver Yices [16]. The maximization of the variable x_4 was performed by simulating a binary search over an interval. The minimum value achieved in our model is trivially 0 at the starting state. This procedure currently requires numerous calls to the solver. Our implementation along with the models used here are available upon request.

Model: The composed model of PCA pump with faults has roughly 6 modes, 11 parameters, 12 real-valued variables and 4 finite domain variables. Since machine faults are assumed to be transient, a counter `nFaults` is used in the model to store the number of machine faults to be considered in any execution. The initial value of this counter is varied from 0 (no fault) to 4 (four faults happen at non-deterministic time instances).

Table 3. Analysis results showing the influence of user error and machine faults on the predicted maximum drug effect chamber concentration. **nFaults:** Number of transient faults in a trace, **Conc:** maximum concentration achieved in ng/mL and **Time:** model checker time (seconds).

Mistake	nFaults = 0		nFaults = 1		nFaults=2		nFaults=4	
	Conc.	Time	Conc.	Time	Conc.	Time	Conc.	Time
None	6.1	64	5.4	447	5.1	1219	4.8	1218
Mode Selection Err.	3.1	16	3.1	213	2.9	500	2.8	459
Unit Selection Err.	150	78	149	210	132	182	101.2	1474
Weight (lbs as kgs)	13.4	73	12.6	1011	11.5	1306	10.5	2250
Weight (kgs as lbs)	2.7	41	2.6	534	2.4	1492	2.2	1617

Table 4. Analysis over 10 randomly selected prescriptions over the range described in Figure 8. The maximum concentration of reference model is normalized to 1. All figures rounded to one significant digit after decimal point.

Mistake	nFaults = 0			nFaults = 1			nFaults=2			nFaults=4		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
None	1	1	1	1	1	1.1	1	1	1.1	0.9	0.8	1.0
Mode Selection Err.	0.2	.1	4	0.3	0.1	0.4	0.3	.2	.4	.2	.1	.4
Unit Selection Err.	23	14	36	22	10	35	23.4	10	33	14.2	14.2	14.2
Weight (lbs as kgs)	2.0	1.8	2.3	2.2	2.1	2.5	2.2	2.1	2.4	2	1.8	2.3
Weight (kgs as lbs)	.4	.3	.4	.4	.4	.4	.4	.4	.5	.4	.4	.4

Table 3 shows the results of the analysis of various fault combinations in terms of the maximum concentration of drug achieved and the time taken by the model checker to complete the analysis. Comparing the maximum drug concentration of $6.1ng/mL$ predicted for the reference model (no human/machine faults) against the maximum values for other fault combinations yields insights into the effects of various faults. For instance, it is clear that mode selection errors roughly halve the maximum concentration achievable. The presence or absence of machine faults has little or no impact on the maximum. On the other hand, unit errors cause a 23 fold increase in the maximum effect chamber concentration achievable.

Overall, the results predicted by model checking agree well with our qualitative predictions of the outcome of the faults. However, the results are more useful since they can provide quantitative predictions and allow us to compare/rank faults based on the severity of their effects. For instance, the under dosage caused by machine faults is predicted to be less significant than that caused by a mode selection error or a unit error wherein the weight in kgs is treated as if it were in lbs. Note that under doses as well as over doses can have serious consequences, since restoring control of pain after an under dose can be difficult.

Finally, we investigate how the overall trends vary with the prescription data. Table 4 summarizes the results of repeating the experiment with 10 different randomly chosen prescriptions. The range of values for various parameters is described in Figure 8. In each case, the maximum value found for the reference execution is normalized to 1, thus allowing us to compare the results from different prescriptions. Table 4 shows the result of this comparison on 10 different randomly generated prescriptions. We note that the overall direction of the effect of error seems independent of the actual prescription data themselves. Furthermore, in many cases, the magnitudes are quite similar.

7 Related Work and Conclusions

A vast body of work has focused on modeling and verification of functional correctness properties for user interfaces (Cf. [14, 33, 34], for instance). Much of this work has focused on the search for specific topologies in the state diagram of the interface that can indicate the presence of serious faults such as *mode confusion* [25, 29, 38]. The problem of characterizing various forms of human error has been studied in detail by cognitive scientists and human factors experts [15, 36]. In this regard, the use of generic mistake models to transform a given ideal interaction into a faulty interactions has been proposed in the past, notably by Hollnagel [22]. Fields presents an integration of this approach with formal verification tools to model and reason about operator error [19].

Formal and Semi-formal techniques have been applied to analyze infusion pumps for the presence of hidden modes and inconsistencies in the interface transitions [44, 45]. Bolton and Bass present a detailed model of the user interaction with an infusion pump by extending the operator function model framework. Their model lends itself to analysis using formal verification tools [6]. Bolton et al. apply their approach to provide a detailed model of the interface and various user actions required to program an infusion. However, in this paper, we consider a detailed model of the pump's operation and the dynamics of the drug concentration in the patient while summarizing the effect of user's mistakes using a mistake model. A combination of the two approaches seems quite desirable. However, it is quite likely that the analysis of such a detailed combined model may be intractable.

Model-based techniques remain popular in many domains including the development of reliable and safe user interfaces [10, 11, 32]. A common approach to the model-based testing of UI applications requires the construction of *mental models* of the user to characterize ways in which the interface can be used [10, 11]. However, these approaches seek to test the functional correctness of the interface itself, while operator errors are usually ignored.

In conclusion, we present a framework for dependability analysis of infusion pumps. In the future, we wish to extend this framework to study other types of programming errors in devices such as implantable pace-makers and total intravenous anesthesia. The model results also support exploration of user interface modifications that could guard against likely errors. One approach could be to incorporate logic into the pump controller to recognize possible user mistakes that may result in underdoses or overdoses. Since the appropriate dose depends on patient weight, so that the doses appropriate for adults are quite different from those for children, for example, it is not possible to specify definitely what the allowable range of doses is, within wide limits. Another suggestion could be to build closed loops to prevent mishaps by estimating drug concentrations using pharmacokinetic models and observable signals such as the respiration rate and blood oxygen content.

References

1. Anonymous (Alberta, R.N.). Lack of standard dosing methods contributes to i.v. infusion errors. Institute for Safe Medication Practices (ISMP) Medication Alert, 64(4) (April 2008)
2. Arney, D., Jetley, R., Jones, P., Lee, I., Sokolsky, O.: Formal methods based development of a PCA infusion pump reference model: Generic infusion pump (GIP) project. In: Proc. High Confidence Medical Devices, Software Systems and Medical Device Plug and Play Interoperability (2007)
3. Arney, D.E., Jetley, R., Jones, P., Lee, I., Ray, A., Sokolsky, O., Zhang, Y.: Generic infusion pump hazard analysis and safety requirements: Version 1.0, CIS Technical Report, University of Pennsylvania. (2009), http://repository.upenn.edu/cis_reports/893 (accessed May 2011)
4. Bates, D.W., Vandervreen, T., Seger, D., Yamaga, C., Rothschild, J.: Variability in intravenous medical practices: Implications for medication safety. J. Joint Commission on Accredication of Healthcare Organizations 31(4), 203–210 (2005)
5. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without bDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
6. Bolton, M.L., Bass, E.J.: Formally verifying human-automation interaction as part of a system model: limitations and tradeoffs. Innovations Syst. Softw. Eng. 6, 219–231 (2010)
7. Bozzano, M., Cimatti, A., Tapparo, F.: Symbolic fault tree analysis for reactive systems. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 162–176. Springer, Heidelberg (2007)
8. Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V.Y., Noll, T., Roveri, M.: The COMPASS approach: Correctness, modelling and performability of aerospace systems. In: Buth, B., Rabe, G., Seyfarth, T. (eds.) SAFECOMP 2009. LNCS, vol. 5775, pp. 173–186. Springer, Heidelberg (2009)
9. Brady, J.L.: First, do no harm: Making infusion pumps safer. Biomedical Instrumentation & Technology 44(5), 372–380 (2010)
10. Brooks, P.A., Memon, A.M.: Automated GUI testing guided by usage profiles. In: Prof. ASE 2007, pp. 333–342. IEEE Press, Los Alamitos (2007)
11. Chinnapongse, V., Lee, I., Sokolsky, O., Wang, S., Jones, P.: Model-based testing of GUI-driven applications. In: Lee, S., Narasimhan, P. (eds.) SEUS 2009. LNCS, vol. 5860, pp. 203–214. Springer, Heidelberg (2009)
12. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
13. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
14. Degani, A., Heymann, M.: Formal Verification of Human-Automation Interaction. Human Factors 44(1), 28–43 (2002)
15. Dekker, S.: The Field Guide to Understanding Human-Error. Ashgate Publishing (2006)
16. Dutertre, B., de Moura, L.: The YICES SMT solver. Cf., http://yices.csl.sri.com/tool-paper.pdf (last viewed January 2009)
17. Ebeling, C.E.: Introduction to Reliability and Maintainability Engineering. Waveland Inc. (2005)
18. Egan, T., Lemmens, H., Fiset, P., Hermann, D., Muir, K., Stanski, D., Shafer, S.: The pharamcokinetics of the new short acting opioid remifentanil (G187084B) in healthy adult male volunteers. Anesthesiology 74, 881–892 (1996)
19. Fields, R.: Analysis of erroneous actions in the design of critical systems. PhD thesis, University of York (January 2001)

20. Grissinger, M.: Misprogram a PCA pump? it's easy!, July 2004. ISMP Medication Safety Alert. (accessed May 2011)
21. Henzinger, T.A.: The theory of hybrid automata. In: LICS 1996, pp. 278–292. IEEE, Los Alamitos (1996)
22. Hollnagel, E.: Human Reliability Analysis Context and Control. Computer And People Series. Academic Press Inc., San Diego (1993)
23. Hollnagel, E.: Cognitive Reliability and Error Analysis Method. Elsevier, Institutt for Energiteknikk, Halden, Norway (1998)
24. Institute for Safe Medication Practices Canada. Fluorocil incident root-cause analysis (2007), http://www.cancerboard.ab.ca/NR/..
25. Joshi, A., Miller, S.P., Heimdahl, M.P.: Mode confusion analysis of a flight guidance system using formal methods. In: 22nd IEEE Digital Avionics Systems Conference, DASC 2003 (October 2003)
26. Kallen, A.: Computational Pharmacokinetics. Chapman & Hall, Boca Raton (2007)
27. Kirwan, B.: A Guide to Practical Human Reliability Assessment. Taylor & Francis, Abington (1994)
28. Lesar, T.S.: Errors in the useof medication dosage equations. Archives of Pediatric Adoloscent Medicine 152, 340–344 (1998)
29. Leveson, N.G., Palmer, E.: Designing automation to reduce operator errors. In: IEEE Trans. on Systems, Man, and Cybernetics, p. 7 (October 1997)
30. McClain, D.A., Hug, C.C.: Intravenous fentanyl kinetics. Clinical Pharmacology & Therapeutics 28(1), 106–114 (1980)
31. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to DPLL(*T*). J. ACM 53(6), 937–977 (2006)
32. Paiva, A., Faria, J.C.P., Tillmann, N., Vidal, R.F.A.M.: A model-to-implementation mapping tool for automated model-based GUI testing. In: Lau, K.-K., Banach, R. (eds.) ICFEM 2005. LNCS, vol. 3785, pp. 450–464. Springer, Heidelberg (2005)
33. Palanque, P.: Formal Methods in Human-Computer Interaction. Springer-Verlag New York, Inc, Heidelberg (1997) ISBN 3540761586
34. Paternó, F., Santoro, C.: Integrating model checking and HCI tools to help designers verify user interface properties. In: Paternó, F. (ed.) DSV-IS 2000. LNCS, vol. 1946, pp. 135–150. Springer, Heidelberg (2001)
35. Pérez-Castañeda, G., Aubry, J.-F., Brinzei, N.: Stochastic hybrid automata model for dynamic reliability assessment. Journal of Risk and Reliability 225(1), 28–41 (2011)
36. Reason, J.T.: Human Error. Cambridge University Press, Cambridge (1990)
37. Rothschild, J., Keohane, C., Cook, E., Orav, E., Burdick, E., Thompson, S., Hayes, J., Bates, D.: A controlled trial of smart infusion pumps to improve medication safety in critically ill patients. Critical care medicine 33(3) (2005)
38. Rushby, J.: Using model checking to help discover mode confusions and other automation surprises. In: Proc. HESSD 1999 (June 1999)
39. Sartori, V., Schumacher, P.M., Bouillon, T., Luginbuehl, M., Morari, M.: On-line estimation of propofol pharamacodynamic parameters. In: Proc. Conference on Engineering in Medicine and Biology, pp. 74–77. IEEE Press, Los Alamitos (2005)
40. Schein, J., Hicks, R., Nelson, W., Sikirica, V., Doyle, D.: Errors in the postoperative period: Causes and prevention. Drug Safety 32(7), 549–559 (2009)
41. Shafer, S.L., Siegel, L.C., Cooke, J.E., Scott, J.C.: Testing computer-controlled infusion pumps by simulation. Anesthesiology 68, 261–266 (1988)
42. Siu, N.: Risk assessment for dynamic systems: An overview. Reliability Engineering & System Safety 43(1), 43–73 (1994)

43. Tabuada, P.: Verification and Control of Hybrid Systems: A Symbolic Approach. Springer, Heidelberg (2009)
44. Thimbleby, H.: Ignorance of interaction programming is killing people. ACM Interactions, 52–57 (2008)
45. Thimbleby, H.: Contributing to safety and due diligence in safety-critical interactive systems development. In: ACM SIGCHI, EICS 2009, pp. 221–230 (2009)
46. Thimbleby, H.: Is it a dangerous prescription? BCS Interfaces 84, 5–10 (2010)
47. Trbovich, P.L., Pinkney, S., Cafazzo, J.A., Easty, A.: The impact of traditional and smart pump infusion technology on nurse medication administration performance in a simulated inpatient unit. Qual. Saf. Health Care 19, 430–434 (2010)
48. Vuyk, J., Mertens, M.J., Olofsen, E., Burm, A.G., Bovill, J.G.: Propofol anesthesia and rational opioid selection. Anesthesiology 87(6), 1549–2562 (1997)

A Design-for-Verification Framework for a Configurable Performance-Critical Communication Interface

Suleiman Abu Kharmeh, Kerstin Eder, and David May

Computer Science Department, University of Bristol,
Woodland Road, United Kingdom
{kharmeh,eder,dave}@cs.bris.ac.uk
http://www.cs.bris.ac.uk

Abstract. In this paper we present a Design-for-Verification framework for a Configurable Performance-Critical Communication Interface. To manage the inherent complexity of the problem we decomposed the interface into independent parametrisable communication blocks. *Tock-CSP* was then used to model the timing and functional specifications of our interface. The FDR model checker and its *tau-priority* model were used to prove that the properties of the configured interface are within the properties of targeted communication protocols.

Keywords: CSP, *tock-CSP*, Design-for-Verification, *tau-priority*.

1 Introduction

Communication interfaces are an important aspect of modern processors especially in the context of multi-core heterogeneous platforms. End users expect I/O interfaces to support a variety of existing protocols [3, 6, 8, 11, 19]. Such interfaces must also be generic to accommodate new protocols. These interfaces are expected to implement a relatively complex set of functional and timing aspects of the target protocols and thereby reduce the communication and processing demands on the processor or network-on-chip.

In practice this can be achieved by providing the user with a set of configuration modes to customise an I/O interface towards a particular type of communication; a complete protocol is thus characterised by a set of configurations of the I/O interface. Generic I/O interfaces are a challenge for both designers and verification engineers. To manage their inherent complexity a structured approach is considered most tractable. This involves decomposing the I/O interface into functionally independent blocks.

These blocks can be configured to obtain a specific instance that matches the functional or timing behaviour required to implement a particular protocol. The parallel composition of these blocks, suitably configured, then provides the overall functional and timing properties of the I/O protocol to be supported. As these blocks can be designed and verified individually at block level this gives rise to a Design-for-Verification approach.

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 335–351, 2011.

This paper demonstrates how such a generic I/O interface can be formalised in Hoare's Communicating Sequential Processes algebra, CSP [9]. CSP was well suited since each functional block can be composed as an independent sequential process and the set of blocks (now CSP processes) synchronize over communication events. We demonstrate how selected example protocols [6, 8] can be obtained by configuring this generic I/O interface. Using the refinement models of CSP a proof is obtained which confirms that the configured instances of the hardware are indeed valid implementations of the respective higher level protocol specifications. We are particularly interested in the specification and automatic model-checking of real-time aspects [21] of the configured system with respect to higher level protocol timing specifications. The *tau-priority* model initially suggested by Ouaknine [18] was recently released in the Failures-Divergence Refinement (FDR) model-checker [7]. Its availability was an additional motivation for the use of CSP.

While the individual ideas of formal protocol specification, hardware decomposition into functional blocks, conformance checking of protocols, and timing requirement validation are well established techniques [3, 11, 12, 22–24], we believe that our integrated formal framework for the design and model-checking of communication protocols' functional and performance aspects is a novel and unique approach. It represents a sound contribution to the ever increasing interest in the verification of real-time aspects of parallel communication system. We also believe that our work presents the first application of the new *tau-priority* CSP model.

1.1 Related Work

The design and formal verification of configurable hardware that can be used for implementing real-time communication protocols has many aspects of related work.

Addressing the model checking aspect of protocols, Seidel [22] has used *tock-CSP* and FDR to model-check the PI-Bus [17] protocol. The work only addressed a specific protocol and does not give any framework for modelling other protocols. It also used an experimental version of FDR's *tau-priority* model and we have been unable to verify the results of that study.

Müffke [16] has presented a framework for the modelling and specification of communication protocols. His work does not address the verification of the protocols specification or the hardware implementation of such protocols.

Böhm and Melham [5] have presented a refinement approach to designing and verifying communication protocols. It addresses on-chip protocols with special focus on the AMBA [4] Bus architecture. The core design step and the following transformations are only relevant to the protocol in question and much of the work requires manual proofs. Applying such a framework to other protocols would require a considerable expertise and effort. It also does not address real-time aspects of protocol specification and verification.

In our previous short paper [1] we presented a brief progress update of our work. At that stage some properties and functional requirements of protocols were investigated. Complete protocols were not yet expressible. In this paper we present the full modelling framework, the Configurable Interface hardware and also the verification framework for specifying and verifying communication protocols.

1.2 Common Communication Protocols' Features

As part of this project a number of communication protocols have been reviewed [3, 6, 8, 11, 17, 19]. A brief discussion of some of those protocols and their functional and timing requirements follows:

- *Universal Asynchronous Receiver/Transmitter (UART)* [6] is a basic protocol any configurable interface is expected to support. It is an asynchronous protocol and there is no external clock exchanged between devices. The implementing interface is expected to have its own time management including an internal clock which is used to sample the data. The protocol is serial in nature and ideally the interface must be able to collect a number of the incoming data bits and present them to the host as one unit possibly stripping out control information such as start/stop bits and parity bits.
- *Serial Peripheral Interface (SPI)* is a de-facto standard protocol that is used between ASIC chips and controller systems. It has no official standard, however, an application note is provided for reference [8]. The main features of the protocol is that it is a master-slave architecture. A clock is exchanged between devices which is generated by the master and used by all slave devices in the system.
- *Media Independent Interface (MII)* [3] is a physical interface used to communicate with Ethernet physical layer chips. This protocol is becoming an essential requirement of an embedded system especially in consumer electronics devices. The bus size is four bits wide and the interface is expected to be able to serialise and deserialise multiple data items while stripping out control information such as the data preamble. Framing signals are exchanged between devices which specify when the data bus carries valid data. In a receiver the interface is expected to use such external signals to enable/disable sampling the data bus and the transmitter interface is expected to auto generate these signals.

From the review the following common features in many protocols were identified:

- The clocking mechanisms and timing requirements are a dominant aspect of all reviewed protocols whether the clock was internally managed or externally exchanged.
- The ability of an external event to automatically trigger internal action in the interface independent of any interaction from the host controller.
- The ability to switch the function of an interface between input and output.

The set of functional and timing observations above were used as a guide for designing the generic I/O interface. This generic configurable interface was later used as a model for checking the full communication protocols' specification.

1.3 Approaches to Hardware Support of Configurability

A brief discussion of the different approaches to supporting interface configurability is given next. The aim is to show the recent trends in the design of communication interfaces, and the need for a generic framework for the design and verification of such interfaces.

Traditionally, hardware protocols were supported using dedicated hardware interfaces with basic configurable options to accommodate the various options within the same protocol [15]. A fundamental functional change to the interface behaviour was impossible and supporting multiple interfaces would require considerable design and verification effort. The development of the FPGA [2] technology on the other hand introduced a true sense of configurability to hardware design. A fully configurable interface was more tangible [10]. Nevertheless this approach suffers from high power consumption and relatively low clocking speeds [13]. May et al. [14] suggested a different approach to configurability which involves providing the user with a communication interface that has a set of fine grained configurability options. The problem with this approach is that it is not easily verifiable using traditional approaches due to the state space explosion problem resulting from the large number of possible configurations.

We devise a framework for the modelling and verification of a configurable communication interface which provides a natural progression to the evolution denoted above. The framework suggested particularly addresses the state space explosion problem through the separation of functional and timing aspects.

2 Formal Modelling: Theory, Concepts and Tools

We briefly discussed in Section 1 the motivation for using CSP. Here we expand on the ideas behind using CSP and *tock-CSP* in particular, and using an example, present the motivation for the use of the *tau-priority* model.

2.1 Tock-CSP

One important aspect of communication protocols is their performance and real-time behaviour. *Tock-CSP* is the discrete approach to timed-CSP designed for specifying and validating performance aspects of protocols automatically. In this approach the passage of time is represented by an external event (usually called *tock*) which is a global event that happens regularly and forever. This changes the semantics of many CSP constructs and models including the *failures/divergences* and *traces* model [18].

Consider this scenario: since no process in a parallel composition should be allowed to run faster than another, all processes must be forced to synchronise on *tock*. Consequently, processes are allowed to be idle and hence pass the *tock* event infinitely often. However, for many semantic reasons including the avoidance of false divergences (i.e. ones where only the *tock* event is involved), some events must have priority over *tock*.

A minimal approach for supporting *tock-CSP* is currently implemented in [7] in which internal events have priority over *tock* under the *tau-priority* model.

Let us consider the following two processes as an example:

$$P1 = (a \rightarrow b \rightarrow tock \rightarrow P1) \setminus a$$
$$P2 = (a \rightarrow b \rightarrow tock \rightarrow P2 \; \Box \; tock \rightarrow P2) \setminus a$$

One way to interpret the difference between the two processes is that process $P2$ takes into account an idle state where event a is not available in which case it can allow *tock* events to happen. $P1$ however must always be able to execute event a immediately at the start which makes event a an *urgent* event. The details of this model can be found in [18, 20].

Under the *tau-priority* model the two processes are understood to be equivalent because, when run independently, both processes are able to execute the event a at the start. Consequently, the choice to execute the *tock* event in process $P2$ is ignored. However, the situation becomes more interesting when those processes are composed in parallel in a wider system in which event a is not available at the start. In this case, process $P1$ would introduce what is called a $time-stop$ in which *tock* events cannot happen. The process $P2$ does not suffer such limitation.

This is useful in the context of communication systems and signal propagation: all signals in the system need to propagate and settle before the next *tock* event and processes which do not have any signal change in that clock cycle should be able to execute the *tock* event gracefully. There are many timing related assertions and checks to be considered when validating such models which will be discussed later in Section 4.

We believe that the *tock-CSP* framework is very promising. It integrates the modelling and verification of real-time performance aspects of embedded systems with the well understood CSP model-checking framework. This paper adds an important contribution through the application of such approach to a complex performance-critical communication interface.

2.2 Abstractions and Refinements and Equivalences

The CSP model-checking framework provides a robust platform for verifying the functional aspects of our design. It allows the specification of a model on various abstraction levels where a model is specified with minimum details. Such specification serves as a top-level specification. Then, by adding various implementation details, we can automatically verify that the added details still conform to the top-level specification by proving a suitable refinement relation of both.

The Evaluation Section 4 provides more details where the refinement relations of various specification and implementation models of both the communication interface as well as the communication protocols in question are checked. Further, we can use the automatic refinement checks to prove that two models are equivalent when we can prove that each model is a refinement of the other. This involves running two refinement checks and is particularly useful for proving that the configured communication system behaves exactly as the top-level protocol specification requires.

3 Formal Modelling: The CSP Models

First a formal abstraction of the individual communication protocols is briefly presented. The need for such an abstraction will become apparent in the evaluation section. Then the formal model of the communication interface is presented. Finally an *Instruction Set Architecture (ISA)-oriented Specification* of the communication protocols is realised which is also further discussed in the evaluation section. The detailed CSP models are beyond the scope of this paper. The code is available online at `http://www.cs.bris.ac.uk/~kharmeh/cspCode.tar` and was verified using FDR version 2.91.

3.1 The Protocol Abstract Specification

An abstract specification of communication protocols is independent of any implementation details. Each protocol can be specified at various abstraction levels. We are most interested in a specification that captures all timing and functional aspects.

To provide timing information at the specification level we deploy *tock-CSP* in a rather simplistic manner: only one process is responsible for executing the *tock* events (called *SpecTimer*). All other processes in this specification synchronise with the *SpecTimer* on events other than *tock* (Reset, Set, Trigger among others). This results in an implicit synchronisation on *tock* events and consequently provides essential timing information for the rest of the specification.

Another process (*Phy* in Figure 2) is devised to provide the specification with an abstract description of the physical interface. Its main function is to register the physical value of the interface set by a transmitter and make this value available to a receiver at any time.

Below we discuss the abstract specification of two simple yet representative protocols which we later use to demonstrate the conformance checking of the Communication Interface. The specification of each protocol is split into a transmitter process and a receiver process which, once connected, form an abstract communication channel.

UART. Different abstraction levels were investigated where timing and functional details were added incrementally. We use an abstraction in which all timing information is present.

Based on system specific assumptions the number of *tock* events per bit (*ClkPerBit*) is calculated.

This is then used to trigger events in the *SpecTimer* process. Such events are then used to control the timing of transmission/reception.

The transmitter (*UartSpecTx*) is modelled as a sequence of events on a CSP channel called *UartOut* which represents changes to the physical transmission line. The transmission process is then defined as an infinite transmission of data words of specific length. The transmission of a data item (*word*) starts with the start bit (*UartOut!0*) followed by a timer Reset. Then, using the *SpecTimer*, all data bits are outputted serially (*UartOut!(word%2)*) in a timely manner. Finally, the transmission ends with the stop bit (*UartOut!1*).

Similarly, the receiver (*UartSpecRx*) waits for the start bit to appear (*UartIn?0*) at which point it sets the *SpecTimer* with duration *1.5 * ClkPerBit*. This is to skip the start bit and sample the first data bit at the halfway point. It does so for all subsequent data bits and observes the stop bit to end reception.

SPI. This protocol has a more complex interface than *UART*. Devices are classified into "master" which is responsible for generating the bus clock (*SCLK*) and "slave" which uses that clock to synchronise its actions on the bus. This external clock should not be confused with the internal system clock events (*tock*).

Our main focus is the timing specification hence we assume there is only one slave in the system. This allows us to abstract away the chip-select line and also assume that the bus is active all the time and that in each clock cycle the data lines carry valid data.

Both Master and Slave devices have two data lines: Master Output Slave Input (*MOSI*) and Master Input Slave Output (*MISO*) and hence both Master and Slave are capable of transmitting and receiving data at the same time.

The number of *tock* events per bus clock cycle is calculated (*ClkPerBit*). According to this the master generates the system clock (*SCLK*) using the *SpecTimer* process described earlier.

The master (*SpiSpecMaster*) initiates the SPI bus cycle by driving the first data bit on the *MOSI* line followed by driving the system clock low (*SCLK!0*). At the half cycle point (at time *ClkPerBit/2*) the clock is driven high (*SCLK!1*) and the data input line is read (*MISO?d*). Finally, at the end of the cycle (at time *ClkPerBit*), the clock is driven low (*SCLK!0*). This is repeated for the number of bits per data item.

The slave bus cycle (*SpiSpecSlave*) is similar to that of the master except that it does not need to generate timing and clock events. Its cycle is controlled by events input on the clock channel (*SCLK*) which are generated by the master.

3.2 The Communication Interface Model

The approach taken to model the communication interface is through the separation of functional requirements into functionally independent configurable hardware blocks. Figure 1 shows a high level view of the different functional blocks modelled.

At this point, four types of configuration modes were modelled: Raw, Conditional, Timed, and Direction.

We also modelled a simple *ISA* to allow us to configure the functional blocks, and consequentially specify the *ISA-oriented Specification* of the communication protocols.

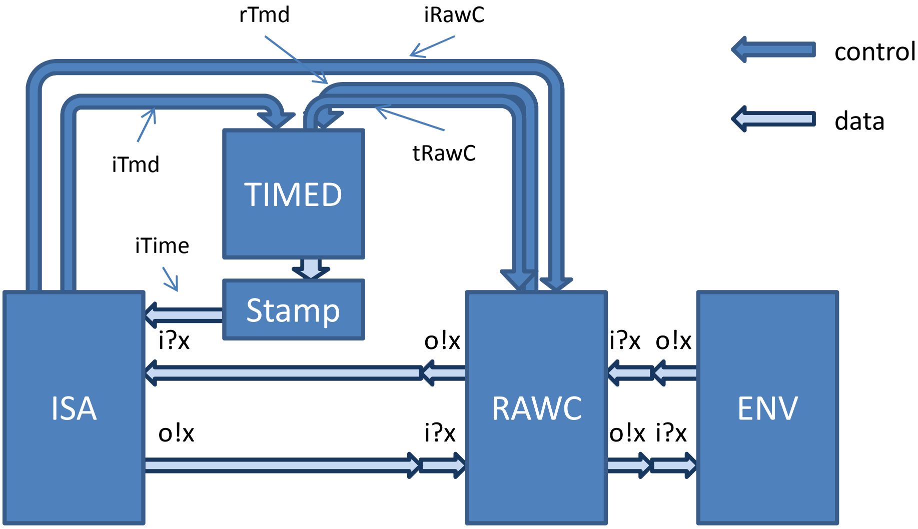


Fig. 1. Communication Interface Model

A description of each functional block and its configurations is given below:

Timed. This functional block is responsible for keeping track of time by keeping a counter of the total number of clock cycles elapsed. This counter has a resolution which is specified by the design engineer. Once this counter reaches its limit, it resets to zero and starts counting up again.

It responds to the following commands:

- Timed I/O (or time dependent I/O): received from the *ISA* block. The *Timed* block compares the required operation time with the current time at each cycle. Once the operation time is reached, it signals the *RawC* block to commit the operation.
- Time-stamping: received from the *RawC* block and their response is internal to the Timed block. It involves the saving of a copy of the current time counter which is called a time-stamp.
- Time-stamp read: received from the *ISA* block. The Timed block responds with the time-stamp value stored at the last executed operation

For the correct timing operation of the system all system components are clocked (i.e. are able to execute *tock* events in an orderly fashion): It is essential for the

RawC block to request the time-stamping at the exact clock cycle the I/O operation completes. It is also essential that ISA instructions happen in a timely manner for the whole system to meet protocol deadlines. Special attention needed to be paid to avoid race conditions and time lags between different system components. The absence of such issues was essential for the system level verification described later.

RawC. This functional block is responsible for interaction with the outside world. It receives I/O commands from the *ISA* block as well as the *Timed* block.

These commands include:

- Unconditional I/O: the *RawC* block can receive this command from the *Timed* block once the correct time of the operation has elapsed. Unconditional I/O can also be initiated from the *ISA* block. The *RawC* block responds back to the *ISA* block with the physical interface state as a response to an input operation. An output operation might trigger a Physical Interface Change command described below.
- Physical Interface Change: the physical interface is modelled as signal change events received and sent by the *RawC* block. The state of the physical interface is kept internally within this block. A signal change event might be sent by this module as a result of an output operation. Alternatively it can be received as a result of an external change to the physical interface triggered by another entity connected to the interface.
- Conditional I/O (or data dependent I/O): the *ISA* block can issue a conditional input command where the condition requires the data on the physical interface to have some specific value. The *RawC* block would only commit this command when the value on the physical interface matches the value requested by the *ISA*.

As mentioned earlier *RawC* needs to synchronise on the timing event *tock* because it is responsible for triggering the timestamps at the exact clock cycle the operation was performed. Many *RawC* processes are then composed in parallel to reflect the number of ports in the system.

ISA. This block is an abstraction of a simple controller instruction set. The modelled instructions are mainly used for communication interface configuration and data I/O. This module synchronises on *tock* events so as to meet communication protocols' deadlines and to avoid many race conditions. For that reason instructions must execute in a timely manner and each instruction takes at least one cycle (one *tock* event). A special No-Operation (NOP) instruction is provided for implementing time delays.

The instructions modelled are:

- Unconditional I/O: takes exactly one clock cycle to execute and does not rely on any timing or data condition to commit.
- Conditional I/O (or data dependent I/O): takes at least one clock cycle and can possibly take infinitely many cycles depending on the physical interface state.

- Timed I/O (or time dependent I/O): takes at least one clock cycle and commits once the specified time is met by the *Timed* block.
- Time-stamp read: takes exactly one clock cycle in which the *Timed* block returns the time-stamp of the last committed operation. This is used to schedule subsequent timed operations.
- No-Operation: takes exactly one cycle. It is useful for implementing time delays.

A top-level abstract description of the *ISA* block is seen as an infinite sequence of any combination of the above instructions.

A communication protocol's *property-oriented specification* [21] is then seen as a well-defined combination of those instructions. For example, a *UART* receiver starts with a *Conditional I/O* to capture the start bit followed by a *Timed I/O* for each data bit. The time of each *Timed I/O* is realised using the *Time-stamp read* instruction. Finally, one last *Timed I/O* is performed to capture the stop bit. The full sequence of instructions for the *UART* receiver is called *UartIsaRx* while the transmitter is called *UartIsaTx*. We call these specifications "*ISA-oriented Specification*".

The detailed specifications of the *SPI* slave (*SpiIsaSlave*) and and *SPI* master (*SpiIsaMaster*) are too complex to be included in this paper

Finally, the whole communication interface is expressed as the parallel composition of all the individual functional blocks:

$$CommInterface = ((ISA \underset{aTimed}{\|} Timed) \underset{aRawC}{\|} RawC)$$

where *aTimed* and *aRawC* are the synchronisation alphabets of the processes *Timed* and *RawC* respectively.

There are many parameters that affect the overall complexity and state space of the communication interface. For example, the number of ports in the system and the size of the time stamp (*TimeSize*). Please refer to Table 1 for more information.

Table 1. Communication Interface Complexity

Port Size	System		ISA		Timed		RawC	
& Time Size	States	Trans	States	Trans	States	Trans	States	Trans
1	880	1840	64	96	54	144	20	62
2	16624	53296	74	114	332	880	400	1860
3	347104	1501984	94	150	2328	6048	8000	50600
4	8191936	44490176	134	222	17456	44608	160000	1290000

We address the complexity issues later in the evaluation section.

4 Evaluation

We demonstrate the framework for the design and formal verification of a configurable communication interface through various progressions of functional and performance checks. First we perform basic functional and timing checks. Then we check the interface conformance incrementally starting from Instruction-level functional checks through system-level functional and performance checks. Finally we check the conformance of the overall interface with targeted communication protocols' abstract specifications.

4.1 Basic Functional and Timing Verification

First, all individual functional blocks as well as the individual protocols' abstract specifications are checked for deadlock and divergence freedom. The divergence and deadlock checks for the timed processes are coupled with basic timing checks:

- Infinitely many ordinary events can never happen between two *tock* events
- No time-stops can happen in a timed process: a time-stop is when the *tock* event cannot happen due to an ill-defined *tock* synchronisation.

The above is true if for each timed process Q the following assertion holds:

$$TOCKS \sqsubseteq_{\mathrm{FD}} Q \setminus (\Sigma \setminus \{tock\})$$

where $TOCKS = tock \rightarrow TOCKS$.

This was used to determine the temporal validity of our models.

4.2 Instruction-Level Functional Verification

We establish the functional correctness of an instruction or a sequence of instructions through what we called earlier an *ISA-oriented Specification*. It involved specifying a sequence of CSP events (a trace) that described a specific functional task (say $UartIsaTx$). We then use this specification to check the unconfigured communication interface's willingness to execute such task (or set of tasks) as follows:

$$CommInterface \sqsubseteq_{\mathrm{FD}} UartIsaTx \setminus \{|consume|\}$$

The $\{|consume|\}$ hidden set of events is used in later checks. Similar checks are performed for all *ISA-oriented Specifications*.

4.3 System-Level Functional Verification

We verify the ability of the CSP models to establish a channel of communication by setting up a system of transmitters and receivers joined together back-to-back. We then check this set-up for equivalence against the functional view of the system as a multi-entry buffer as described in Section 4.3 of Roscoe [20]. This is performed for both the protocols' abstract specifications 3.1 as well as the Communication Interface's *ISA-oriented Specification* 3.2. This establishes the functional correctness of the transmitters' and receivers' specifications and establishes that data integrity is maintained through the communication process.

For example, an *SPI* specification channel composed of the *SpiSpecMaster* and *SpiSpecSlave* is called *SpiSpecChannel* as in Figure 2.

If we let *SpiSpecSlave=SSS*, *SpiSpecMaster=SSM*, *SpiSpecChannel=SSC* and *aPhy=*{*i1, o1, i2, o2*} then:

$$SSC = \left(SSS[\![^{i1,\,i2}/_{miso,\,mosi}]\!] \underset{i1,i2}{\|} Phy(aPhy) \right) \underset{o1,o2,sclk}{\|} SSM[\![^{o1,\,o2}/_{miso,\,mosi}]\!]$$

We then define the process *Buffer* to be a multi-entry buffer with alphabet $aBuffer = \{|produce, consume|\}$.

A delayable buffer $DelBuf = Buffer \mid\mid\mid TOCKS$ is one which allows the passage of any number of clock cycles between input and output. This is useful for the *tau-priority* refinement model in FDR version 2.91 where events are either delayable or urgent. We use this delayable buffer to verify the functional correctness of the *SpiSpecChannel* as follows:

$$DelBuf \sqsubseteq_{\mathrm{T}} SSC \setminus \Sigma \setminus (aBuffer \cup \{tock\}) : [tau\, priority\, over] : \{tock\}$$

On the right-hand side of the refinement all events apart from $aBuffer \cup \{tock\}$ are made internal. It is necessary to make the *tock* event visible and subsequently give it lower priority with respect to internal events under the *tau-priority* model.

Similar checks are performed to verify the functional correctness of *UartSpecTx* and *UartSpecRx*. Using the same approach, two similar channels connecting *UartIsaTx* to *UartIsaRx* and *SpiIsaMaster* to *SpiIsaSlave* were constructed as illustrated in Figure 3. The functional equivalence of those channels to a multi-entry buffer was then verified.

This evaluation approach reduces the complexity of the overall system considerably because the number of all possible configurations is reduced to those which are essential to the functional correctness of the system as specified. Table 2 provides more details.

Table 2. Evaluation Models Complexity

Time Size	*SpiIsaMaster*		*SpiIsaSlave*		*SpiIsaChannel*	
	States	Trans	States	Trans	States	Trans
2	5547	11798	4433	11669	4986	8601
3	12603	26614	11161	29461	10258	17369
4	31323	65462	31529	83477	22146	36249
5	87195	180022	99913	265237	51298	79385

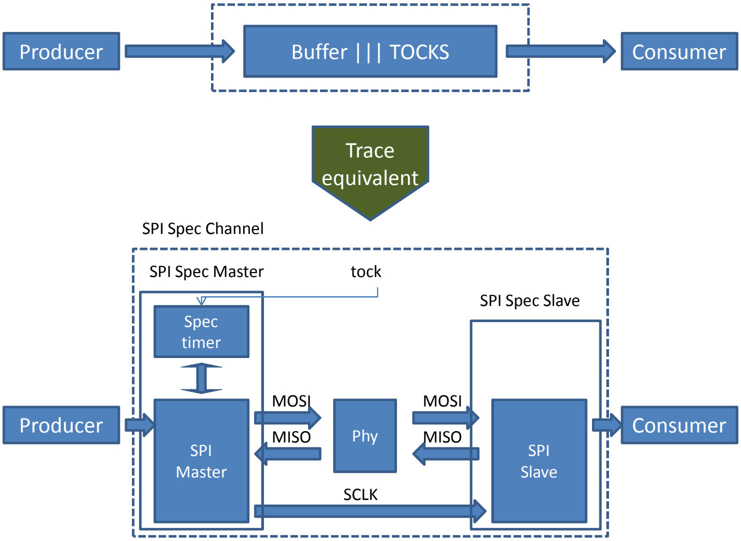


Fig. 2. SPI Specification Channel Verification

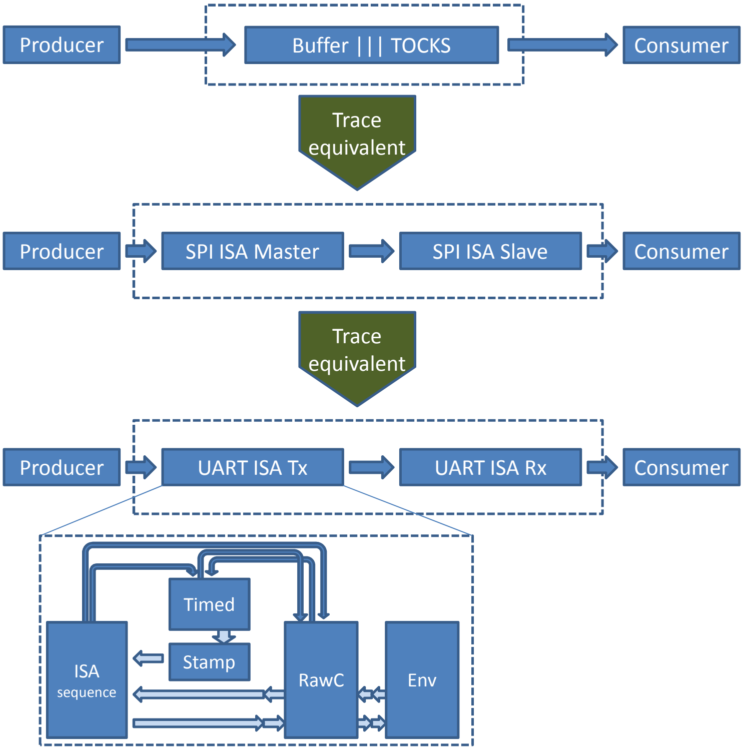


Fig. 3. ISA-oriented Channels Verification

4.4 System-Level Performance Verification

Performance metrics are possible to verify automatically through translating those metrics into a *tock-CSP* specification process.

For instance, a latency trace specification could be:

$$\begin{aligned} TOCK(0) &= SKIP \\ TOCK(n) &= \mathit{tock} \rightarrow TOCK(n-1) \\ LAT(N) &= \mathit{produce?byte} \rightarrow TOCK(N)\, ;\, \mathit{consume!byte} \rightarrow SKIP \\ &\quad \square\ \mathit{tock} \rightarrow LAT(N) \end{aligned}$$

If we know that the latency of a data word through the *SpiSpecChannel* to be X then we can use the latency specification above to verify the *SpiSpecChannel* performance:

$$LAT(X) \sqsubseteq_{\mathrm{T}} \mathit{SpiSpecChannel} \setminus \Sigma \smallsetminus (aBuffer \cup \{\mathit{tock}\})$$

It can be tricky to translate performance metrics into trace specifications especially when one does not know in advance the numerical value for such metric. In such case, a construction can be made to extract this numerical value automatically using FDR (see Section 14.6 of Roscoe [20] for more details).

4.5 Protocol Conformance Verification

The final set of checks performed were to establish that the *ISA-oriented Specification* of a protocol is equivalent to the abstract specification of that protocol. For example, *UartSpecRx* and *UartIsaRx* are equivalent and interchangeable. We prove this through the functional verification of a channel construction where at one end of the channel is a transmitter's *ISA-oriented Specification* (*UartIsaTx*) and the other end is a receiver's abstract protocol specification (*UartSpecRx*). Let *UartIsaTx*=*UIT*, *UartSpecRx*=*USR* and $aUIT = \{\!| uTx, \mathit{tock}, \mathit{produce} |\!\}$ then the conformance channel (*ConfChan*) is defined as follows:

$$ConfChan = \left(UIT \setminus \Sigma \smallsetminus aUIT \underset{uTx,\mathit{tock}}{\|} Phy(uTx, uRx) \right) \underset{uRx,\mathit{tock}}{\|} USR$$

The refinement check:

$$DelBuf \sqsubseteq_{\mathrm{T}} ConfChan \setminus \Sigma \smallsetminus (aBuffer \cup \{\mathit{tock}\}) : [\mathit{tau\,priority\,over}] : \{\mathit{tock}\}$$

verifies that the *ISA-oriented Specification* of a transmitter is compatible with (and hence conforms to) the abstract protocol specification of a receiver. Similar checks can be constructed to verify the conformance of other functional and performance aspects of the protocols.

The total number of automatic assertion checks performed was 51. To avoid any memory limitations we use the 64 bit version of FDR and we run it on a machine with 40GB physical memory. We also use compression techniques built into FDR to reduce verification time. By setting the *TimeSize* to 4 bits FDR spent 31 minutes to verify all assertions.

5 Conclusion and Outlook

To manage the inherent complexity of a configurable performance-critical communication interface, we demonstrated a Design-for-Verification framework for such an interface. This is achieved through the separation of functional aspects into independent hardware building blocks with clear interfaces and functions which can be verified at the block level.

We took a novel approach to protocol conformity checking, whereby a protocol's abstract specification was verified with respect to an instance *configuration* of our Communication Interface's CSP model through what we called *ISA-oriented Specification*. These conformity checks carried out in Section 4.5 not only confirm protocol functional conformance but also timing conformance.

The integration of performance and functional modelling in the general CSP framework is a promising approach. This is achieved through the use of *tock-CSP* and the *tau-priority* model to carry out timing analysis and performance checks at an early stage of the design cycle. This provides crucial and precise real-time metrics of the system as well as performance conformance verification with respect to a particular protocol. Our project demonstrates this approach, and we believe it is the first project to use the newly released *tau-priority* model.

The complexity and grand scale of recent design problems especially in multicore parallel systems with multiple clock domains highlight the importance of formal design and verification and the automation of its tools. Though FDR proved very useful for designing systems and protocols in the past, it requires a good knowledge of CSP and its theories. This is one of the reasons for the lack of wide adoption in the design community. With the addition of the *tau-priority* model it becomes even more important to address the usability of FDR. For that reason, and as part of an ongoing work, we are investigating improvements to the user interface of FDR such as the auto-generation of timing waveforms as well as Finite State Machine graphs. Such improvements can be useful for analysing hardware and timing specifications. This would greatly improve the adoption of the tool in general as well as any new design approaches such as *tock-CSP* and the *tau-priority* model. We are also investigating the reduction of the verification time by parallel optimisation of the front end of FDR.

Future work also involves the investigation of other functional and performance aspects of hardware design. Our target is to re-use the developed framework to model and verify more complex interfaces with additional functional blocks and multiple clock domains. We also aim to target communication protocols of higher functional and performance requirements.

Acknowledgements. The authors would like to thank Oxford Computer Laboratory and Formal Systems (Europe) Ltd for providing the FDR2 model checking tool and for providing much needed technical support for the tool and its CSPm interface.

References

[1] AbuKharmeh, S., Eder, K., May, D.: Formal analysis of a programmable performance-critical processor communication interface. In: Proceedings of the 10th International Workshop on Automated Verification of Critical Systems (2010), http://www.cs.bris.ac.uk/Publications/pub_master.jsp?id=2001286

[2] Altera Corporate: Stratix GX FPGA, Family Datasheet (2004)

[3] American National Standards Institute: Standards for Local Area Networks: Carrier Sense Multiple Access with Collision Detection (CSMA-CD) - Standards 802.3. John Wiley & Sons, Inc., New York (1985)

[4] ARM Ltd.: AMBA$^{\text{TM}}$Specification Revision 2 (1999)

[5] Böhm, P., Melham, T.: A refinement approach to design and verification of on-chip communication protocols. In: Proceedings of the 2008 International Conference on Formal Methods in Computer-Aided Design (2008)

[6] Electronics Industries Association: EIA standard RS-232-C interface between data terminal equipment and data communication equipment employing serial data interchange. Tech. rep., Electronics Industries Association (1969)

[7] Formal Systems: Failures-Divergence Refinement FDR2, User Manual, 2.91 edn. (May 2010)

[8] Freescale Semiconductor Inc: MPC5121e serial peripheral interface (SPI). online (2009)

[9] Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International, Englewood Cliffs (2004)

[10] Horta, E.L., Lockwood, J.W., Taylor, D.E., Parlour, D.: Dynamic hardware plugins in an FPGA with partial run-time reconfiguration. In: Proceedings of the 39th Annual Design Automation Conference, DAC 2002, vol. (39), pp. 343–348. ACM, New York (2002)

[11] International Organization for Standardization: ISO 11898-1:2003 - Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling. Tech. rep., ISO (2003)

[12] Kaizhi, Y.: Validating system requirements by functional decomposition and dynamic analysis. In: Proceedings of the 11th International Conference on Software Engineering, ICSE 1989, pp. 188–196. ACM, New York (1989)

[13] Kuon, I., Rose, J.: Measuring the gap between FPGAs and ASICs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 12(8), 203–215 (2007)

[14] May, D., Muller, H., Hedinger, P., Dixon, A., Owen, R., Richards, N.: XS1 Ports: use and specification. XMOS Ltd., 1.02 edn. (November 2008)

[15] Microchip Inc: PIC16F87X Microcontrollers Datasheet (2001)

[16] Müffke, F.: A better way to design communication protocols. Ph.D. thesis, University of Bristol (May 2004)

[17] Open Microprocessor Systems Initiative: OMI 324: PI-Bus. Tech. rep., OMI (1994)

[18] Ouaknine, J.: Discrete analysis of continuous behaviour in real-time concurrent systems. Ph.D. thesis, Oxford University (2001)

[19] Philips Semiconductors: The I^{2}C-Bus specification. Tech. rep., Philips (2000)

[20] Roscoe, A.W.: Understanding Concurrent Systems, 1st edn. Texts in Computer Science. Springer, Heidelberg (2010)

[21] Schneider, S.: Concurrent and Real Time Systems: The CSP Approach. John Wiley & Sons, Inc., New York (1999)

[22] Seidel, K.: Case study: Specification and refinement of the PI-Bus. In: Naftalin, M., Bertrán, M., Denvir, T. (eds.) FME 1994. LNCS, vol. 873, pp. 532–546. Springer, Heidelberg (1994)

[23] Spars, J., Furber, S.: Principles of Asynchronous Circuit Design: A Systems Perspective. Kluwer Academic Publishers, Boston (2001)

[24] Tretmans, J.: A Formal Approach to Conformance Testing. Ph.D. thesis, University of Twente, Enschede (1992)

Author Index