Jean-François Raskin
P.S. Thiagarajan (Eds.)

Formal Modeling and Analysis of Timed Systems

LNCS 4763

5th International Conference, FORMATS 2007
Salzburg, Austria, October 2007
Proceedings

Springer

Lecture Notes in Computer Science 4763

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Jean-François Raskin P.S. Thiagarajan (Eds.)

Formal Modeling and Analysis of Timed Systems

5th International Conference, FORMATS 2007
Salzburg, Austria, October 3-5, 2007
Proceedings

Springer

Volume Editors

Jean-François Raskin
Université Libre de Bruxelles, Computer Science Department
Campus de la Plaine, CP 212, Boulevard du Triomphe, 1050 Brussels, Belgium
E-mail: jraskin@ulb.ac.be

P.S. Thiagarajan
National University of Singapore, School of Computing
Computing 1, Law Link, Singapore 117590, Republic of Singapore
E-mail: thiagu@comp.nus.edu.sg

Library of Congress Control Number: 2007935932

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

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

ISSN 0302-9743
ISBN-10 3-540-75453-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-75453-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12169450 06/3180 5 4 3 2 1 0

Preface

This volume consists of the proceedings of the Fifth International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2007). The main goal of this series of conferences is to bring together diverse communities of researchers that deal with the timing aspects of computing systems. Both fundamental and practical aspects of timed systems are addressed and results reporting new application domains are actively encouraged. Further, invited talks that survey various aspects of this broad research domain were presented at the conference.

FORMATS 2007 was co-located (during October 3–5) as a guest conference at the Embedded Systems Week, which constitutes a week-long event that brings together conferences, tutorials and workshops dealing with various aspects of embedded systems research and development. Embedded Systems Week took place this year at Salzburg, Austria during September 30 - October 5, 2007. Detailed information about FORMATS 2007 can be found at http://www.ulb.ac.be/di/formats07, while http://www.esweek.org provides an overview of the Embedded Systems Week Event. We would like to thank the organizers of the Embedded Systems Week for enabling FORMATS 2007 to be co-located at this exciting event and for providing valuable logistics support.

This year we received 48 submissions and the Programme Committee selected 22 submissions for presentation at the conference. FORMATS 2007 used the EasyChair conference system to manage the reviewing process. The topics dealt with by the accepted papers cover: the theory of timed and hybrid systems, analysis and verification techniques, case studies and novel applications. We wish to thanks the Programme Committee members and the other reviewers for their competent and timely review of the submissions. We also wish to sincerely thank the three invited speakers, Franck Cassez, Joost-Pieter Katoen, and Bruce Krogh, for accepting our invitation and providing extended abstracts of their talks to be included in the proceedings.

As always, the Springer LNCS team provided excellent support in the preparation of this volume. Finally, our heartfelt thanks are due to Martin De Wulf, who put in a great deal of work towards the compilation of these proceedings.

July 2007

P.S. Thiagarajan
Jean-François Raskin

Organization

Programme Chairs

Jean-François Raskin (ULB, Belgium)
P.S. Thiagarajan (NUS, Singapore)

Programme Committee

Rajeev Alur(University of Pennsylvania, USA)
Eugène Asarin (University of Paris 7, France)
Patricia Bouyer (LSV, France)
Ed Brinksma (ESI Eindhoven, The Netherlands)
Véronique Bruyère (UMH, Belgium)
Flavio Corradini (University of Camerino, Italy)
Goran Frehse (Verimag, France)
Martin Fränzle (University of Oldenbourg, Germany)
Salvatore La Torre (University of Salerno, Italy)
Insup Lee (University of Pennsylvania, USA)
Rupak Majumdar (University of California, Los Angeles, USA)
Nicolas Markey (LSV, France)
Brian Nielsen (University of Aalborg, Denmark)
Joël Ouaknine (University of Oxford, UK)
Paritosh Pandya (TIFR-ITT Bombay, India)
Paul Pettersson (University of Mälardalen, Sweden)
Mariëlle Stoelinga (University of Twente, The Netherlands)
Stavros Tripakis (Cadence Berkeley Labs, USA)
Frits Vaandrager (University of Nijmegen, The Netherlands)

External Reviewers

Yasmina Abdeddaim
Madhukar Anand
David Arney
Samalam Arun-Kumar
Louise Avila
Grégory Batt
Danièle Beauquier
Gerd Behrmann
Albert Benveniste
Jasper Berendsen
Jan Bergstra
Dragan Bosnacki
Ahmed Bouajjani
Hichem Boudali
Marius Bozga
Tomas Brazdil
Thomas Brihaye
Diletta Cacciagrano
Thomas Chatain
Ling Cheung

Fabrice Chevalier
Ricardo Corin
Pieter Cuijpers
Adrian Curic
Pedro R. D'Argenio
Deepak D'Souza
Alexandre David
Conrado Daws
Francesco De Angelis
Martin De Wulf
Jerry den Hartog
Maria Rita Di Berardini
Henning Dierks
Catalin Dima
Nikhil Dinesh
Laurent Doyen
Arvind Easwaran
Andreas Eggers
Abraham Erika
Marco Faella
Ansgar Fehnker
Bernd Finkbeiner
Hans-Joerg Peter
Sebastian Fischmeister
Olga Grinchtein
Christian Herde
Holger Hermanns
Jane Hillston
John Håkansson
Sumit Jha
Joost-Pieter Katoen
Xu Ke
Pavel Krcal
Moez Krichen
Morten Kühnrich
Rom Langerak
Ruggero Lanotte
François Laroussinie
Martin Leucker
Oded Maler
Jasen Markovski
Marius Mikucionis
Aniello Murano
Margherita Napoli
Wonhong Nam
Dejan Nickovic
Brian Nielsen
Mimmo Parente
Anders Pettersson
Linh Phan
Barbara Re
Arend Rensink
Pierre-Alain Reynier
Oliviero Riganelli
Julien Schmaltz
Cristina Seceleanu
Mihaela Sighireanu
Jiri Srba
Vijay Suman
Daniel Sundmark
Mani Swaminathan
Li Tan
Tino Teige
Luca Tesei
Tayssir Touili
Marina Waldén
Jim Weimer
James Worrell

Table of Contents

Abstraction of Probabilistic Systems

Joost-Pieter Katoen[1,2]

[1] RWTH Aachen University, Software Modeling and Verification Group, Germany
[2] University of Twente, Formal Methods and Tools, The Netherlands

1 Introduction

Probabilistic model checking enjoys a rapid increase of interest from different communities. Software tools such as PRISM [13] (with about 4,000 downloads), MRMC [12], and LiQuor [2] support the verification of Markov chains or variants thereof that exhibit nondeterminism. They have been applied to case studies from areas such as randomised distributed algorithms, planning and AI, security, communication protocols, biological process modeling, and quantum computing. Probabilistic model checking engines have been integrated in existing tool chains for widely used formalisms such as stochastic Petri nets [6], Statemate [5], and the stochastic process algebra PEPA [11], and are used for a probabilistic extension of Promela [2].

The typical kind of properties that can be checked is time-bounded reachability properties—"Does the probability to reach a certain set of goal states (by avoiding bad states) within a maximal time span exceed $\frac{1}{2}$?"—and long-run averages—"In equilibrium, does the likelihood to leak confidential information remain below 10^{-4}?" Extensions for cost-based models allow for checking more involved properties that refer to e. g., the expected cumulated cost or the instantaneous cost rate of computations. Intricate combinations of numerical or simulation techniques for Markov chains, optimisation algorithms, and traditional LTL or CTL model-checking algorithms result in simple, yet very efficient verification procedures. Verifying time-bounded reachability properties on models of tens of millions of states usually is a matter of seconds.

Like in the traditional setting, probabilistic model checking suffers from the *state space explosion* problem: the number of states grows exponentially in the number of system components and cardinality of data domains. To combat this problem, various techniques from traditional model checking have been adopted such as binary decision diagrams (multi-terminal BDDs) [10], partial-order reduction [8] and abstract interpretation [14]. We will focus on bisimulation minimisation for fully probabilistic models such as discrete-time and continuous-time Markov chains (DTMCs and CTMCs, for short), and variants thereof with costs. They are an important class of stochastic processes that are widely used in practice to determine system performance and dependability characteristics.

We first study the comparative semantics of branching-time relations for fully probabilistic systems. Strong and weak (bi)simulation relations are covered together with their characterisation in terms of probabilistic and continuous-time variants of CTL, viz. the temporal logics PCTL [9] and CSL[1,3]. PCTL is a

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 1–3, 2007.

discrete-probabilistic variant of CTL in which existential and universal path quantification have been replaced by a probabilistic path operator. CSL includes in addition means to impose time-bounds on (constrained) reachability problems. For instance, it allows one to stipulate that the probability of reaching a certain set of goal-states within a specified real-valued time bound, provided that all paths to these states obey certain properties, is at least/at most some probability value. The result of this study [4] is an overview of weak and strong (bi)simulations relations, including connections between discrete- and continuous-time relations.

In particular, strong probabilistic bisimulation preserves the validity of PCTL and CSL formulas. It implies ordinary *lumpability*, an aggregation technique for Markov chains that is omnipresent in performance and dependability evaluation since the 1960s. Quotient Markov chains can be obtained in a fully automated way. The time complexity of quotienting is logarithmic in the number of states, and linear in the number of transitions—as for traditional bisimulation minimisation—when using splay trees (a specific kind of balanced tree) for storing blocks [7]. Experimental results show that—as for traditional model checking—enormous state space reductions (up to logarithmic savings) may be obtained. In contrast to traditional model checking, in many cases, the verification time of the original Markov chain exceeds the quotienting time plus the verification time of the bisimulation quotient. This effect is stronger for bisimulations that are tailored to the property to be checked and applies to PCTL as well as CSL model checking.

Finally, we present a more aggressive abstraction technique for DTMCs and CTMCs that uses a three-valued interpretation, i.e., a formula evaluates to either true, false or indefinite. Abstract DTMCs, in fact Markov decision processes (MDPs), are obtained by replacing transition probabilities by intervals where lower and upper bounds act as under- and over-approximation, respectively. For CTMCs, we resort to uniform CTMCs, i.e., CTMCs in which all states have equal residence times and use transition probability intervals. Any CTMC can be efficiently turned into a weak-bisimilar uniform CTMC. Abstraction then amounts to just replace probabilistic transitions by intervals, and model checking can be reduced to determining (constrained) time-bounded reachability probabilities in continuous-time MDPs. This abstraction is conservative for affirmative and negative verification results and allows to perform abstraction on models where bisimulation fails.

Acknowledgement. Thanks to my co-workers on these topics: Christel Baier, Holger Hermanns, David N. Jansen, Tim Kemna, Daniel Klink, Verena Wolf, and Ivan S. Zapreev.

References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous time Markov chains. ACM TOCL 1, 162–170 (2000)
2. Baier, C., Ciesinski, F., Größer, M.: ProbMela and verification of Markov decision processes. Performance Evaluation Review 32, 22–27 (2005)

3. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: Model-checking algorithms for continuous-time Markov chains. IEEE TSE 29, 524–541 (2003)
4. Baier, C., Katoen, J.-P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for Markov chains. Information and Computation 200, 149–214 (2005)
5. Böde, E., Herbstritt, M., Hermanns, H., Johr, S., Peikenkamp, T., Pulungan, R., Wimmer, R., Becker, B.: Compositional performability evaluation for Statemate. In: QEST, pp. 167–178. IEEE CS, Los Alamitos (2006)
6. D'Aprile, D., Donatelli, S., Sproston, J.: CSL model checking for the GreatSPN tool. In: Aykanat, C., Dayar, T., Körpeoğlu, İ. (eds.) ISCIS 2004. LNCS, vol. 3280, pp. 543–553. Springer, Heidelberg (2004)
7. Derisavi, S., Hermanns, H., Sanders, W.H.: Optimal state-space lumping in Markov chains. IPL 87, 309–315 (2003)
8. Groesser, M., Baier, C.: Partial order reduction for Markov decision processes: a survey. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2005. LNCS, vol. 4111, pp. 408–427. Springer, Heidelberg (2006)
9. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6, 512–535 (1994)
10. Hermanns, H., Kwiatkowska, M., Norman, G., Parker, D., Siegle, M.: On the use of MTBDDs for performability analysis and verification of stochastic systems. J. of Logic and Alg. Progr. 56, 23–67 (2003)
11. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, Cambridge (1996)
12. Katoen, J.-P., Khattri, M., Zapreev, I.S.: A Markov reward model checker. In: QEST, pp. 243–244. IEEE CS, Los Alamitos (2005)
13. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: a hybrid approach. Int. J. on STTT 6, 128–142 (2004)
14. Monniaux, D.: Abstract interpretation of programs as Markov decision processes. Science of Computer Programming 58, 179–205 (2005)

From Analysis to Design

Bruce H. Krogh

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213, USA
krogh@ece.cmu.edu

Abstract. Research in formal methods has emphasized analysis techniques for system verification and testing. Despite the successful and growing integration of tools using formal methods into production design flows, there is very limited use of formal methods for design per se. This is understandable, given the considerable complexity of synthesis relative to analysis. Direct synthesis may not be the only way formal methods could contribute more significantly to design, however. Most tools used for design are actually analysis tools that have been extended in various ways to provide information useful for design, such as sensitivities from numerical optimization and simulation. Using embedded control systems as an application context, this plenary talk will review how analysis tools, including formal methods, are currently used in the design flow. We will then suggest research directions for strengthening the use of formal methods for design. One approach will be illustrated using extensions to a recently developed abstraction-based method for verifying linear hybrid automata with a large number of continuous state variables.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, p. 4, 2007.

Efficient On-the-Fly Algorithms for Partially Observable Timed Games*

Franck Cassez

CNRS/IRCCyN
1 rue de la Noë
BP 92101
44321 Nantes Cedex 3, France
franck.cassez@cnrs.irccyn.fr
http://www.irccyn.fr/franck

Abstract. In this paper, we review some recent results on the efficient synthesis of controllers for timed systems. We first recall the basics of controller synthesis for timed games and then present an efficient on-the-fly algorithm for reachability games and its extension to partially observable timed games.

The material of this paper is based on two recent articles [13,14] that introduced truly on-the-fly algorithms for the synthesis of controllers for timed games. These results were obtained together with Alexandre David, Emmanuel Fleury and Kim G. Larsen (Aalborg University, Denmark), Didier Lime (IRCCyN, France) and Jean-François Raskin (ULB, Brussels, Belgium).

1 Introduction

The *control problem* (CP) for discrete event systems was first studied by Ramadge & Wonham in [24]. The CP is the following: "Given a finite-state model of a *plant* P (*open system)* with controllable and uncontrollable discrete actions and a *control objective* Φ, does there exist a *controller* f such that the plant supervised by f (*closed system*) satisfies Φ?"

The *dense-time* version of the CP with an *untimed* control objective has been investigated and solved in [23]. In this seminal paper, Maler *et al.* consider a plant P given by a *timed game automaton* which is a standard timed automaton [4] with its set of discrete actions partitioned into controllable and uncontrollable actions. They give an algorithm to decide whether a controller exists or not, and show that if one such controller exists, a witness can be effectively computed. In [28], Wong-Toi has given a semi-algorithm to solve the CP when the plant is defined by an extended class of timed game which is a hybrid (game) automaton.

The algorithms for computing controllers for timed games are based on backwards fix-point computations of the set of winning states [23,7,17]. For timed

* Work supported by the French National Research Agency ANR-06-SETI-DOTS and by the Fonds National de la Recherche Scientifique, Belgium.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 5–24, 2007.

game automata, they were implemented in the tool KRONOS [2] at the end of 90's but lack efficiency because they require the computation of the complete set of winning states. Moreover the backward computation may sometimes not terminate or be very expensive for some extended classes of timed game automata, for instance if integer assignments of the form $i := j + k$ are allowed on discrete transitions.

In the last ten years, a lot of progress has been made in the design of efficient tools for the analysis (model-checking) of timed systems. Tools like KRONOS [12] or UPPAAL [21] have become very efficient and widely used to check properties of timed automata but still no real efficient counterpart had been designed for timed games.

One of the reason may be that on-the-fly algorithms have been absolutely crucial to the success of these model-checking tools. Both reachability, safety as well as general liveness properties of such timed models may be decided using on-the-fly algorithms *i.e.* by exploring the reachable state-space in a symbolic forward manner with the possibility of early termination. Timed automata technology has also been successfully applied to optimal scheduling problems with on-the-fly algorithms which quickly lead to near-optimal (time- or cost-wise) schedules [6,5,18,25,1].

Regarding timed games, in [27,3], Altisen and Tripakis have proposed a partially on-the-fly method for solving timed games. However, this method involves an extremely expensive preprocessing step in which the quotient graph of the timed game *w.r.t.* time-abstracted bisimulation[1] needs to be built. Once obtained this quotient graph may be used with any on-the-fly game-solving algorithms for untimed systems.

In a recent paper [13], we have proposed an efficient, truly on-the-fly algorithm for the computation of winning states for (reachability) timed game automata. Our algorithm is a symbolic extension of the on-the-fly algorithm suggested by Liu & Smolka in [22] for linear-time model-checking of finite-state systems. Being on-the-fly, this symbolic algorithm may terminate before having explored the entire state-space, *i.e.* as soon as a winning strategy has been identified. Also the individual steps of the algorithm are carried out efficiently by the use of so-called zones as the underlying data structure.

This algorithm has been implemented in UPPAAL-TIGA [8] which is an extension of the tool UPPAAL [21]. Some recent experiments with UPPAAL-TIGA are reported in [13] and show promising results. More recently in [14], we have extended this algorithm to deal with partially observable timed games and implemented it in a prototype based on UPPAAL-TIGA.

In this paper we focus on *reachability timed games* and present the on-the-fly algorithms of [13,14] and conclude with some current research directions.

The plan of the paper is the following: in Section 2 we recall the basics of timed game automata and the backwards algorithms used to compute safety and reachability games. In Section 3 we present the efficient truly on-the-fly

[1] A time-abstracted bisimulation is a binary relation on states preserving discrete states and abstracted delay-transitions.

algorithm for reachability games that was introduced in [13] and implemented in UPPAAL-TIGA. In Section 4 we show how it can be adapted [14] to compute winning states for timed games under partial observation. Finally in Section 5 we give some current research directions.

2 Backward Algorithms for Solving Timed Games

Timed Game Automata (TGA) were introduced in [23] for solving control problems on timed systems. This section recalls the basic results for the controller synthesis for TGA. For a more complete survey the reader is referred to [10].

2.1 Notations

Let X be a finite set of real-valued variables called clocks. $\mathbb{R}_{\geq 0}$ stands for the set of non-negative reals. We note $\mathcal{C}(X)$ the set of constraints φ generated by the grammar: $\varphi ::= x \sim k \mid x-y \sim k \mid \varphi \wedge \varphi$ where $k \in \mathbb{Z}$, $x, y \in X$ and $\sim \in \{<, \leq, =, >, \geq\}$. $\mathcal{B}(X)$ is the subset of $\mathcal{C}(X)$ that uses only rectangular constraints of the form $x \sim k$. A *valuation* v of the variables in X is a mapping $v : X \to \mathbb{R}_{\geq 0}$. We let $\mathbb{R}^X_{\geq 0}$ be the set of valuations of the clocks in X. We write $\mathbf{0}$ for the valuation that assigns 0 to each clock. For $Y \subseteq X$, we denote by $v[Y]$ the valuation assigning 0 (*resp.* $v(x)$) to any $x \in Y$ (*resp.* $x \in X \setminus Y$). We denote $v + \delta$ for $\delta \in \mathbb{R}_{\geq 0}$ the valuation *s.t.* for all $x \in X$, $(v + \delta)(x) = v(x) + \delta$. For $g \in \mathcal{C}(X)$ and $v \in \mathbb{R}^X_{\geq 0}$, we write $v \models g$ if v satisfies g and $[\![g]\!]$ denotes the set of valuations $\{v \in \mathbb{R}^X_{\geq 0} \mid v \models g\}$. A *zone* Z is a subset of $\mathbb{R}^X_{\geq 0}$ *s.t.* $[\![g]\!] = Z$ for some $g \in \mathcal{C}(X)$.

2.2 Timed Automata and Simulation Graph

Definition 1 (Timed Automaton [4]). *A* Timed Automaton *(TA) is a tuple* $A = (L, \ell_0, \mathsf{Act}, X, E, \mathsf{Inv})$ *where* L *is a finite set of* locations, $\ell_0 \in L$ *is the* initial *location,* Act *is the set of* actions, X *is a finite set of* real-valued clocks, $E \subseteq L \times \mathcal{B}(X) \times \mathsf{Act} \times 2^X \times L$ *is a finite set of* transitions, $\mathsf{Inv} : L \to \mathcal{B}(X)$ *associates with each location its* invariant.

A *state* of a TA is a pair $(\ell, v) \in L \times \mathbb{R}^X_{\geq 0}$ that consists of a location and a valuation of the clocks. From a state $(\ell, v) \in L \times \mathbb{R}^X_{\geq 0}$ *s.t.* $v \models \mathsf{Inv}(\ell)$, a TA can either let time progress or do a discrete transition. This is defined by the transition relation $\longrightarrow \subseteq (L \times \mathbb{R}_{\geq 0}) \times \mathsf{Act} \cup \mathbb{R}_{\geq 0} \times (L \times \mathbb{R}_{\geq 0})$ built as follows:

- for $a \in \mathsf{Act}$, $(\ell, v) \xrightarrow{a} (\ell', v')$ if there exists a transition $\ell \xrightarrow{g,a,Y} \ell'$ in E *s.t.* $v \models g$, $v' = v[Y]$ and $v' \models \mathsf{Inv}(\ell')$;
- for $\delta \geq 0$, $(\ell, v) \xrightarrow{\delta} (\ell, v')$ if $v' = v + \delta$ and $v, v' \in [\![\mathsf{Inv}(\ell)]\!]$.

Thus the semantics of a TA is the labeled transition system $S_A = (Q, q_0, \mathsf{Act} \times \mathbb{R}_{\geq 0}, \longrightarrow)$ where $Q = L \times \mathbb{R}^X_{\geq 0}$, $q_0 = (\ell_0, \mathbf{0})$ and the set of labels is $\mathsf{Act} \cup \mathbb{R}_{\geq 0}$. A *run* of a timed automaton A is a (finite or infinite) sequence of alternating

time and discrete transitions in S_A. We use $\mathsf{Runs}((\ell, v), A)$ for the set of runs that start in (ℓ, v). We write $\mathsf{Runs}(A)$ for $\mathsf{Runs}((\ell_0, \mathbf{0}), A)$. If ρ is a finite run we denote $last(\rho)$ the last state of the run. An example of a timed automaton is given in Figure 1 where $[x \leq 4]$ denotes the invariant of location ℓ_4.

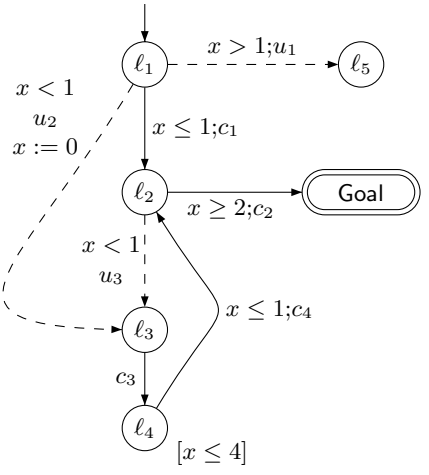


Fig. 1. A Timed Game Automaton

The analysis of TA is based on the exploration of a graph, the *simulation graph*, where the nodes are *symbolic states*. A symbolic state is a pair (ℓ, Z) where $\ell \in L$ and Z is a zone of $\mathbb{R}^X_{\geq 0}$. Let $S \subseteq Q$ and $a \in \mathsf{Act}$ we define the a-successors and a-predecessors of S respectively by:

$$\mathsf{Post}_a(S) = \{(\ell', v') \mid \exists (\ell, v) \in S, (\ell, v) \xrightarrow{a} (\ell', v')\}$$
$$\mathsf{Pred}_a(S) = \{(\ell, v) \mid \exists (\ell', v') \in S, (\ell, v) \xrightarrow{a} (\ell', v')\}.$$

The set of timed successors, $S^{\nearrow}$, of S is defined by:

$$S^{\nearrow} = \{(\ell, v + d) \mid (\ell, v) \in S \cap [\![\mathsf{Inv}(\ell)]\!], (\ell, v + d) \in [\![\mathsf{Inv}(\ell)]\!],\ d \in \mathbb{R}_{\geq 0}\}.$$

Let $\Longrightarrow$ be the relation defined on symbolic states by: $(\ell, Z) \overset{a}{\Longrightarrow} (\ell', Z')$ if $(\ell, g, a, Y, \ell') \in E$ and $Z' = ((Z \cap [\![g]\!])[Y])^{\nearrow}$. The simulation graph $SG(A)$ of A is given by the labeled transition system $(Z(Q), S_0, \mathsf{Act}, \Longrightarrow)$, where $Z(Q)$ is the set of zones of Q, $S_0 = ((\{(\ell_0, \mathbf{0})\}^{\nearrow}) \cap [\![\mathsf{Inv}(\ell_0)]\!]$ and $\Longrightarrow$ defined as above. If A is *bounded*, *i.e.* all the clocks are bounded, the number of symbolic states is finite and $SG(A)$ is finite as well. Otherwise, a finite simulation graph that preserves for instance reachability property can be constructed for any TA. In this case, we can either transform the given TA into an equivalent one in which all location-invariants insist on an upper bound on all clocks or, alternatively, we can apply standard extrapolation *w.r.t.* maximal constant occurring in the TA (which is correct up to time-abstracted bisimulation).

2.3 Safety and Reachability Games

Definition 2 (Timed Game Automaton [23]). *A* Timed Game Automaton (TGA) G *is a timed automaton with its set of actions* Act *partitioned into* controllable (Act_c) *and* uncontrollable (Act_u) *actions.*

The automaton in Figure 1 is also a TGA: controllable actions are depicted with plain arrows and uncontrollable ones with dashed arrows.

Given a TGA G and a set[2] of states $K \subseteq L \times \mathbb{R}^X_{\geq 0}$ the *reachability control problem* consists in finding a *strategy* f s.t. G supervised by f enforces G to enter a state in K. The *safety control problem* is the dual asking for the strategy to constantly avoid K. By "a reachability game (G, K)" (*resp.* safety) we refer to the reachability (*resp.* safety) control problem for G and K.

Let (G, K) be a reachability (resp. safety) game. Assume that all the states reachable from $(l, v) \in K$ are also in K. A finite or infinite run $\rho = (\ell_0, v_0) \xrightarrow{e_0} (\ell_1, v_1) \xrightarrow{e_1} \cdots \xrightarrow{e_n} (\ell_{n+1}, v_{n+1}) \cdots$ in $\mathsf{Runs}(G)$ is *winning* if there is some $k \geq 0$ *s.t.* $(\ell_k, v_k) \in K$ (*resp.* for all $k \geq 0$, $(\ell_k, v_k) \in K$). We rule out runs with an infinite number of consecutive time transitions of duration 0. The set of winning runs in G from (ℓ, v) is denoted $\mathsf{WinRuns}((\ell, v), G)$.

The formal definition of the control problems is based on the definitions of *strategies* and *outcomes*. A strategy [23] is a function that during the course of the game constantly gives information as to what the controller should do in order to win the game. In a given situation, the strategy could suggest the controller to either *i*) "do a particular controllable action" or *ii*) "do nothing at this point in time, just wait" which will be denoted by the special symbol λ. Let $G = (L, \ell_0, \mathsf{Act}, X, E, \mathsf{Inv})$ be a TGA and $S_G = (Q, q_0, \rightarrow)$ its semantics.

Definition 3 (Strategy). *A* strategy f *over* G *is a partial function from the finite runs of* $\mathsf{Runs}(G)$ *to* $\mathsf{Act}_c \cup \{\lambda\}$ s.t. *for every finite run* ρ

- *if* $f(\rho) \in \mathsf{Act}_c$ *then* $last(\rho) \xrightarrow{f(\rho)} (\ell', v')$ *for some* (ℓ', v') *and*
- *if* $f(\rho) = \lambda$ *then* $last(\rho) \xrightarrow{\delta} (\ell', v')$ *for some* $\delta > 0$ *and* (ℓ', v').

We denote $\mathsf{Strat}(G)$ the set of strategies over G. A strategy f is *state-based* if $\forall \rho, \rho' \in \mathsf{Runs}(G), last(\rho) = last(\rho')$ implies that $f(\rho) = f(\rho')$. State-based strategies are also called *memoryless* strategies in game theory [17,26].

The restricted behavior of a TGA G controlled with some strategy f is defined by the notion of *outcome* [17].

Definition 4 (Outcome). *Let* f *be a strategy over* G. *The (set of)* outcomes $\mathsf{Outcome}(q, f)$ *of* f *from* q *in* S_G *is the subset of* $\mathsf{Runs}(q, G)$ *defined inductively by:*

- $q \in \mathsf{Outcome}(q, f)$,
- *if* $\rho \in \mathsf{Outcome}(q, f)$ *then* $\rho' = \rho \xrightarrow{e} q' \in \mathsf{Outcome}(q, f)$ *if* $\rho' \in \mathsf{Runs}(q, G)$ *and one of the following three conditions hold:*

[2] For real computation we shall require that K is defined as a finite union of symbolic states.

1. $e \in \mathsf{Act}_u$,
2. $e \in \mathsf{Act}_c$ *and* $e = f(\rho)$,
3. $e \in \mathbb{R}_{\geq 0}$ *and* $\forall 0 \leq e' < e, \exists q'' \in Q$ s.t. $last(\rho) \xrightarrow{e'} q'' \wedge f(\rho \xrightarrow{e'} q'') = \lambda$.

– *for an infinite run* ρ, $\rho \in \mathsf{Outcome}(q, f)$ *if all the finite prefixes of* ρ *are in* $\mathsf{Outcome}(q, f)$.

We assume that uncontrollable actions can only spoil the game and the controller has to do some controllable action to win [7,23,18]. In other words, an uncontrollable action cannot be forced to happen in G. Thus, a run may end in a state where only uncontrollable actions can be taken. For reachability games we assume w.l.o.g. that the goal is a particular location Goal *i.e.* $K = \{(\mathsf{Goal}, v) \,|\, v \in \mathbb{R}^X_{\geq 0}\}$ as depicted on Figure 1. For safety games we have to avoid a particular location Bad *i.e.* $K = \{(\mathsf{Bad}, v) \,|\, v \in \mathbb{R}^X_{\geq 0}\}$.

In the sequel we focus on *reachability games*. A *maximal run* ρ is either an infinite run (supposing no infinite sequence of delay transitions of duration 0) or a finite run ρ that satisfies either *i*) $last(\rho) \in K$ or *ii*) if $\rho \xrightarrow{a}$ then $a \in \mathsf{Act}_u$ *i.e.* the only possible discrete actions from $last(\rho)$ (if any) are uncontrollable actions. A strategy f is *winning* from q if all maximal runs in $\mathsf{Outcome}(q, f)$ are in $\mathsf{WinRuns}(q, G)$. A state q in a TGA G is *winning* if there exists a winning strategy f from q in G. We denote by $\mathcal{W}(G)$ the set of winning states in G and $\mathsf{WinStrat}(q, G)$ the set of winning strategies from q over G.

2.4 Backward Algorithms for Solving Timed Games

Let $G = (L, \ell_0, \mathsf{Act}, X, E, \mathsf{Inv})$ be a TGA. For timed games, the computation of the winning set of states is based on the definition of a *controllable predecessors* operator [17,23]. The controllable and uncontrollable discrete predecessors of $S \subseteq Q$ are defined by $\mathsf{cPred}(S) = \bigcup_{c \in \mathsf{Act}_c} \mathsf{Pred}_c(S)$ and $\mathsf{uPred}(S) = \bigcup_{u \in \mathsf{Act}_u} \mathsf{Pred}_u(S)$. A notion of *safe* timed predecessors of a set S *w.r.t.* a set U is also needed. Intuitively a state q is in $\mathsf{Pred}_t(S, U)$ if from q we can reach $q' \in S$ by time elapsing and along the path from q to q' we avoid U. This operator is formally defined by:

$$\mathsf{Pred}_t(S,U) = \{q \in Q \mid \exists \delta \in \mathbb{R}_{\geq 0} \ s.t. \ q \xrightarrow{\delta} q', q' \in S \text{ and } \mathsf{Post}_{[0,\delta]}(q) \subseteq \overline{U}\} \quad (1)$$

where $\mathsf{Post}_{[0,\delta]}(q) = \{q' \in Q \mid \exists t \in [0, \delta] \ s.t. \ q \xrightarrow{t} q'\}$ and $\overline{U} = Q \setminus U$. The *controllable predecessors* operator π is formally defined as follows (Figure 2):

$$\pi(S) = \mathsf{Pred}_t\left(S \cup \mathsf{cPred}(S), \mathsf{uPred}(\overline{S})\right) \quad (2)$$

Note that according to this definition, even forced uncontrollable actions (*e.g.* by an invariant) are not bound to happen and cannot help to win. A controllable action must be taken to reach a winning state and uncontrollable actions can only spoil the game.

If S is a finite union of symbolic states, then $\pi(S)$ is again a finite union of symbolic states and $\pi(S)$ is effectively computable. Assume (G, K) is a reachability game. In this case the least fix-point of $S = K \cup \pi(S)$ can be computed

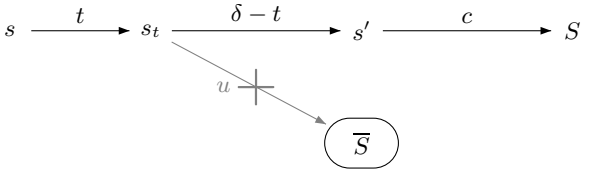


Fig. 2. $\pi(S)$

by the iterative process given by $W^0 = K$ and $W^{n+1} = W^n \cup \pi(W^n)$. This computation will converge after finitely many steps for TGA [23] and the we denote W^* the fix-point. As it is proved in [23], $W^* = \mathcal{W}(G)$. Note that W^* is the maximal (complete) set of winning states of G *i.e.* a state is winning iff it is in W^*. Thus there is a winning strategy in G iff $(\ell_0, \mathbf{0}) \in W^*$. Altogether this gives a symbolic algorithm for solving reachability games. For safety games, it suffices to take the greatest fix-point W^* of $S = K \cap \pi(S)$ and again $W^* = \mathcal{W}(G)$.

Another important result for reachability and safety TGA is that memoryless strategies are sufficient to win [7,23]. This makes it possible to compute a most permissive state-based strategy. Extracting strategies can be done using the set of winning states W^* (see [9] for reachability games).

For the example of Figure 1, the set of symbolic winning states is given by: $\mathcal{W} = \{(\ell_1, x \leq 1), (\ell_2, x \leq 2), (\ell_3, x \leq 1), (\ell_4, x \leq 1), (\mathsf{Goal}, x \geq 0)\}$.

A winning strategy would consist in taking c_1 immediately in all states (ℓ_1, x) with $x \leq 1$; taking c_2 immediately in all states (ℓ_2, x) with $x \leq 2$; taking c_3 immediately in all states (ℓ_3, x) and delaying in all states (ℓ_4, x) with $x < 1$ until the value of x is 1 at which point the edge c_4 is taken.

3 On-the-Fly Algorithm for Reachability Games

For finite-state systems, on-the-fly model-checking algorithms has been an active and successful research area since the end of the 80's, with the algorithm proposed by Liu & Smolka [22] being particularly elegant (and optimal).

In [13], we have proposed a version of this algorithm for timed games. In the sequel we first present the on-the-fly algorithm for reachability *untimed* games and then show how to design a symbolic version for *timed* games.

3.1 On-the-Fly Algorithm for Discrete Games

Untimed games are a restricted class of timed games with only finitely many states Q and with only discrete actions, *i.e.* the set of labels in the semantics of the game is Act. Hence (memoryless) strategies amounts to choosing a controllable action given the current state, *i.e.* $f : Q \to \mathsf{Act}_c$. For (untimed) reachability games we assume a designated location Goal and the purpose of the analysis is to decide the existence of a strategy f where all runs contains Goal.

The on-the-fly algorithm, OTFUR, we have proposed in [13] is given in Fig. 3. The idea for this algorithm is the following: we assume that some variables store two sets of transitions: *ToExplore* store the transitions that have explored

and *ToBackPropagate* store the transitions the target states of which has been declared winning. Another variable, *Passed*, stores the set of states that have already been encountered. Each encountered state $q \in Passed$ has a status, $Win[q]$ which is either *winning* (1) or *unknown* (0). We also use a variable $Depend[q]$ that stores for each q, the set of explored transitions t s.t. q is a target of t. The initial values of the variables are set by lines 2 to 6.

To perform a step of the on-the-fly algorithm OTFUR, we pick transition a (q, α, q') in $ToExplore \cup ToBackPropagate$ (line 10) and process it as follows:

- if the target state q' is encountered for the first time ($q' \notin Passed$), we update *Passed*, *Depend* and $Win[q']$ (lines 14–16). We also initialize some counters (lines 12 and 13) $c(q')$ and $u(q')$ which have the following meaning: at each time, $c(q')$ represents the number of controllable transitions that can be taken to reach a winning state from q' and $u(s)$ represents the number of uncontrollable hazardous transitions from q' *i.e.* those for which we do not know yet if they lead to a winning state. When q' is first encountered $u(q')$ is simply the number of outgoing uncontrollable transitions from q'. Finally (lines 17 to 20), depending on the status of q' we add the outgoing transitions to *ToExplore* or just schedule the current transition for back propagation if q' is winning.
- in case $q' \in Passed$, it means that either its status has been changed recently (and we just popped a transition from *ToBackPropagate*) or that a new transition leading to q' has been chosen (from *ToExplore*). We thus check whether the status of q' is winning and if yes, we update some information on q: lines 24 and 25 updates the counters $c(q)$ or $u(q)$ depending on the type of the transition being processed (controllable or not). The state q can be declared winning (line 27) if at least one controllable transition leads to a winning state ($c(q) \geq 1$) and all outgoing uncontrollable transitions lead to a winning state as well ($u(q) = 0$). In this case the transitions leading to q ($Depend[q]$) are scheduled for back propagation (line 29). Otherwise we have just picked a new transition leading to q' and we only update $Depend[q']$ (line 31).

The correctness proof of this algorithm is given by the following theorem:

Theorem 1 ([13]). *Upon termination of* OTFUR *on a given untimed game G the following holds:*

1. *If $q \in Passed$ and $Win[q] = 1$ then $q \in \mathcal{W}(G)$;*
2. *If $(ToExplore \cup ToBackPropagate) = \emptyset$ and $Win[q] = 0$ then $q \notin \mathcal{W}(G)$.*

In addition to being on-the-fly and correct, this algorithm terminates and is optimal in that it has linear time complexity in the size of the underlying untimed game: it is easy to see that each edge $e = (q, \alpha, q')$ will be added to *ToExplore* at most once and to *ToBackPropagate* at most once as well, the first time q is encountered (and added to *Passed*) and the second time when $Win[q']$ changes winning status from 0 to 1. Notice that to obtain an algorithm running in linear time in the size of G (*i.e.* $|Q| + |E|$) it is important that the reevaluation of the winning status of a state q is performed using the two variables $c(q)$ and $u(q)$.

1: **Initialization**
2: $Passed \leftarrow \{q_0\}$;
3: $ToExplore \leftarrow \{(q_0, \alpha, q') \mid \alpha \in \mathsf{Act}, q \xrightarrow{\alpha} q'\}$;
4: $ToBackPropagate \leftarrow \emptyset$;
5: $Win[q_0] \leftarrow (q_0 = \mathsf{Goal}\ ?\ 1 : 0)$; *// set status to 1 if q_0 is* Goal
6: $Depend[q_0] \leftarrow \emptyset$;
7: **Main**
8: **while** $((ToExplore \cup ToBackPropagate \neq \emptyset)) \wedge Win[q_0] \neq 1))$ **do**
9: *// pick a transition from ToExplore or ToBackPropagate*
10: $e = (q, \alpha, q') \leftarrow pop(ToExplore)$ or $pop(ToBackPropagate)$;
11: **if** $q' \notin Passed$ **then**
12: $c(q') = 0$;
13: $u(q') = |\{(q' \xrightarrow{a} q'', a \in \mathsf{Act}_u\}|$;
14: $Passed \leftarrow Passed \cup \{q'\}$;
15: $Depend[q'] \leftarrow \{(q, \alpha, q')\}$;
16: $Win[q'] \leftarrow (q' = \mathsf{Goal}\ ?\ 1 : 0)$;
17: **if** $Win[q'] = 0$ **then**
18: $ToExplore \leftarrow ToExplore \cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
19: **else**
20: $ToBackPropagate \leftarrow ToBackPropagate \cup \{e\}$;
21: **else**
22: **if** $Win[q'] = 1$ **then**
23: *// update the counters of the state q*
24: **if** $\alpha \in \mathsf{Act}_c$ **then** $c(q) \leftarrow c(q) + 1$;
25: **else** $u(q) \leftarrow u(q) - 1$;
26: *// re-evaluate the status of the state q*
27: $Win[q] \leftarrow (c(q) \geq 1) \wedge (u(q) = 0)$;
28: **if** $Win[q]$ **then**
29: $ToBackPropagate \leftarrow ToBackPropagate \cup Depend[q]$;
30: **else** *// $Win[q'] = 0$*
31: $Depend[q'] \leftarrow Depend[q'] \cup \{e\}$;
32: **endif**
33: **endif**
34: **endwhile**

Fig. 3. OTFUR: **O**n-**T**he-**F**ly Algorithm for **U**ntimed **R**eachability Games

3.2 On-the-Fly Algorithm for Timed Games

We can extend algorithm OTFUR to the timed case using a zone-based forward and on-the-fly algorithm for solving timed reachability games. The algorithm, SOTFTR, is given in Fig. 4 and may be viewed as an interleaved combination of *forward computation* of the *simulation graph* of the timed game automaton together with *back-propagation* of information *of winning states*. As in the untimed case the algorithm is based on two sets, *ToExplore* and *ToBackPropagate*, of symbolic edges in the simulation-graph, and a passed-list, *Passed*, containing all the symbolic states of the simulation-graph encountered so far by the algorithm. The crucial point of our symbolic extension is that the winning

1: **Initialization**
2: $Passed \leftarrow \{S_0\}$ where $S_0 = \{(\ell_0, \mathbf{0})\}^{\nearrow}$;
3: $ToExplore \leftarrow \{(S_0, \alpha, S') \mid S' = \mathsf{Post}_\alpha(S_0)^{\nearrow}\}$;
4: $ToBackPropagate \leftarrow \emptyset$;
5: $Win[S_0] \leftarrow S_0 \cap (\{\mathsf{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
6: $Depend[S_0] \leftarrow \emptyset$;
7: **Main**
8: **while** $((ToExplore \cup ToBackPropagate \neq \emptyset) \wedge (\ell_0, \mathbf{0}) \not\in Win[S_0]))$ **do**
9: *// pick a transition from ToExplore or ToBackPropagate*
10: $e = (S, \alpha, S') \leftarrow pop(ToExplore)$ or $pop(ToBackPropagate)$;
11: **if** $S' \notin Passed$ **then**
12: $Passed \leftarrow Passed \cup \{S'\}$;
13: $Depend[S'] \leftarrow \{(S, \alpha, S')\}$;
14: $Win[S'] \leftarrow S' \cap (\{\mathsf{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
15: **if** $Win[S'] \subsetneq S'$ **then**
16: $ToExplore \leftarrow ToExplore \cup \{(S', \alpha, S'') \mid S'' = \mathsf{Post}_\alpha(S')^{\nearrow}\}$;
17: **if** $Win[S'] \neq \emptyset$ **then**
18: $ToBackPropagate \leftarrow ToBackPropagate \cup \{e\}$;
19: **else**
20: *// If $T \notin Passed$, we assume $Win[T] = \emptyset$*
21: $Good \leftarrow Win[S] \cup \bigcup_{S \xrightarrow{c} T} \mathsf{Pred}_c(Win[T])$;
22: $Bad \leftarrow \bigcup_{S \xrightarrow{u} T} \mathsf{Pred}_u(T \setminus Win[T])) \cap S$;
23: $Win^* \leftarrow \mathsf{Pred}_t(Good, Bad)$;
24: **if** $(Win[S] \subsetneq Win^*)$ **then**
25: $Waiting \leftarrow Waiting \cup Depend[S]$;
26: $Win[S] \leftarrow Win^*$;
27: $Depend[S'] \leftarrow Depend[S'] \cup \{e\}$;
28: **endif**
29: **endwhile**

Fig. 4. SOTFTR: **S**ymbolic **O**n-**T**he-**F**ly Algo. for **T**imed **R**eachability Games.

status of an individual symbolic state is no more 0 or 1 but is now the *subset* $Win[S] \subseteq S$ (union of zones) of the symbolic state S which is currently known to be winning. The set $Depend[S]$ indicates the set of edges (or predecessors of S) which must be reevaluated (*i.e.* added to *ToBackPropagate*) when new information about $Win[S]$ is obtained, *i.e.* when $Win[S] \subsetneq Win^*$. Whenever a symbolic edge $e = (S, \alpha, S')$ is considered with $S' \in Passed$, the edge e is added to the dependency set of S' so that that possible future information about additional winning states within S' may also be back-propagated to S. In Table 1, we illustrate the forward exploration and backwards propagation steps of the algorithm.

The correctness of the symbolic on-the-fly algorithm SOTFTR is given by the theorem:

Theorem 2 ([13]). *Upon termination of the algorithm* SOTFTR *on a given timed game automaton G the following holds:*

1. *If $(\ell, v) \in Win[S]$ for some $S \in Passed$ then $(\ell, v) \in \mathcal{W}(G)$;*

Table 1. Running SOTFTG

Steps			*ToExplore* ∪ *ToBackPropagate*	*Passed*	*Depend*	*Win*
#	S	S'				
0	-	-	$(S_0, u_1, S_1), (S_0, u_2, S_2), (\mathbf{S_0}, \mathbf{c_1}, \mathbf{S_3})$	S_0	-	$(S_0, \emptyset)$
1	S_0	S_3	$(S_0, u_1, S_1), (S_0, u_2, S_2)$ + $(\mathbf{S_3}, \mathbf{c_1}, \mathbf{S_4}), (S_3, u_3, S_2)$	S_3	$S_3 \mapsto (S_0, c_1, S_3)$	$(S_3, \emptyset)$
2	S_3	S_4	$(S_0, u_1, S_1), (S_0, u_2, S_2), (S_3, u_3, S_2)$ + $(\mathbf{S_3}, \mathbf{c_2}, \mathbf{S_4})$	S_4	$S_4 \mapsto (S_3, c_2, S_4)$	(S_4, S_4)
3	S_3	S_4	$(S_0, u_1, S_1), (S_0, u_2, S_2), (S_3, u_3, S_2)$ + $(\mathbf{S_0}, \mathbf{c_1}, \mathbf{S_3})$	-	-	$(S_3, x \geq 1)$
4	S_0	S_3	$(S_0, u_1, S_1), (S_0, u_2, S_2), (\mathbf{S_3}, \mathbf{u_3}, \mathbf{S_2})$	S_4	$S_3 \mapsto (S_0, c_1, S_3)$	$(S_0, x = 1)$
5	S_3	S_2	$(S_0, u_1, S_1), (S_0, u_2, S_2)$ + $(\mathbf{S_2}, \mathbf{c_3}, \mathbf{S_5})$	S_2	$S_2 \mapsto (S_3, u_3, S_2)$	$(S_2, \emptyset)$
6	S_2	S_5	$(S_0, u_1, S_1), (S_0, u_2, S_2)$ + $(\mathbf{S_5}, \mathbf{c_4}, \mathbf{S_3})$	S_5	$S_5 \mapsto (S_2, c_3, S_2)$	$(S_5, \emptyset)$
7	S_5	S_3	$(S_0, u_1, S_1), (S_0, u_2, S_2)$ + $(\mathbf{S_2}, \mathbf{c_3}, \mathbf{S_5})$	-	$S_3 \mapsto$ (S_2, c_3, S_2) (S_5, c_4, S_3)	$(S_5, x \leq 1)$
8	S_2	S_5	$(S_0, u_1, S_1), (S_0, u_2, S_2)$ + $(\mathbf{S_3}, \mathbf{u_3}, \mathbf{S_2})$	-	$S_5 \mapsto (S_2, c_3, S_2)$	$(S_2, x \leq 1)$
9	S_3	S_2	$(S_0, u_1, S_1), (\mathbf{S_0}, \mathbf{u_2}, \mathbf{S_2})$ + $(S_0, c_1, S_3), (S_5, c_4, S_3)$	-	-	(S_3, S_3)
10	S_0	S_2	$(S_0, u_1, S_1), (S_0, c_1, S_3), (\mathbf{S_5}, \mathbf{c_4}, \mathbf{S_3})$	-	$S_2 \mapsto$ (S_3, u_3, S_2) (S_0, u_2, S_2)	$(S_0, x \leq 1)$
11	S_5	S_3	$(S_0, u_1, S_1), (\mathbf{S_0}, \mathbf{c_1}, \mathbf{S_3})$	-	-	-
12	S_0	S_3	$(\mathbf{S_0}, \mathbf{u_1}, \mathbf{S_1})$	-	-	-
13	S_0	S_1	$\emptyset$	S_1	$S_1 \mapsto (S_0, u_1, S_1)$	$(S_1, \emptyset)$

At step n, $(\mathbf{S}, \alpha, \mathbf{S'})$ is the transition popped at step $n+1$;
At step n, $+(S, \alpha, S')$ the transition added to *ToBackPropagate* or *ToExplore* at step n;
Symbolic States: $S_0 = (\ell_1, x \geq 0)$,$S_1 = (\ell_5, x > 1)$, $S_2 = (\ell_3, x \geq 0)$, $S_3 = (\ell_2, x \geq 0)$, $S_4 = (\mathsf{Goal}, x \geq 2)$, $S_5 = (\ell_4, x \geq 0)$

2. *If* $\mathit{ToExplore} \cup \mathit{ToBackPropagate} = \emptyset$, $(\ell, v) \in S \setminus \mathit{Win}[S]$ *for some* $S \in \mathit{Passed}$ *then* $(\ell, v) \notin \mathcal{W}(G)$.

Termination of the algorithm SOTFTR is guaranteed by the finiteness of the number of symbolic states of $SG(A)$. Moreover, each edge (S, α, T) will be present in the *ToExplore* and *ToBackPropagate* at most $1 + |T|$ times, where $|T|$ is the number of regions of T: (S, α, T) will be in *ToExplore* the first time that S is encountered and subsequently in *ToBackPropagate* each time the set $Win[T]$ increases. Now, any given region may be contained in several symbolic states of the simulation graph (due to overlap). Thus the SOTFTR algorithm is *not* linear in the region-graph and hence not theoretically optimal, as an algorithm with linear worst-case time-complexity could be obtained by applying the untimed algorithm directly to the region-graph. However, this is only a theoretical result and it turns out that the implementation of the algorithm in UPPAAL-TIGA is very efficient.

We can optimize (space-wise) the previous algorithm. When we explore the automaton forward, we check if any newly generated symbolic state S' belongs *Passed*. As an optimization we may instead use the classical inclusion check: $\exists S'' \in \mathit{Passed}$ s.t. $S' \subseteq S''$, in which case, S' is discarded and we update $\mathit{Depend}[S'']$ instead. Indeed, new information learned for S'' can be new information on S' but not necessarily. This introduces an overhead (time-wise) in the sense that we may back-propagate irrelevant information. On the other hand, back-propagating only the relevant information would be unnecessarily complex

and would void most of the memory gain introduced by the use of inclusion. In practice, the reduction of the number of forward steps obtained by the inclusion check pays off for large systems and is a little overhead otherwise, as shown in our experiments. It is also possible to propagate information about losing states: in the case of reachability games, if a state is a deadlock state and is not winning, for sure it is losing. This can also speed-up the algorithm. For the example of Figure 1, we can propagate the information that $(\ell_5, x > 1)$ is losing which entails $(\ell_1, x > 1)$ is losing as well. Then it only remains to obtain the status of $(\ell_1, x \leq 1)$ to determine if the game is winning or not.

4 On-the-Fly Algorithm for Partially Observable Games

4.1 Partial Observability

In the previous sections we have assumed that the controller has perfect information about the system: at any time, the controller will know precisely in what state the system is. In general however — e.g. due to limited sensors — a controller will only have imperfect (or partial) information about the state of the environment. For instance some uncontrollable actions may be *unobservable*. In the discrete case it is well known how to handle partial observability of actions as it roughly amounts to determinize a finite state system.

However for the timed case under partial observability of events, it has been shown in [11] that the controller synthesis problem is in general undecidable. Fixing the resources of the controller (i.e. a maximum number of clocks and maximum allowed constants in guards) regains decidability [11], a result which also follows from the quotient and model construction results of [19,20].

Another line of work [16,29] recently revisited partial observability for finite state systems. This time, the partial observability amounts to imperfect information on the state of the system. Only a finite number of possible *observations* can be made on the system configurations and this provides the sole basis for the strategy of the controller. The framework of [16,29] is essentially turn-based. Moreover, the controller can make an observation after each discrete transition. It could be that it makes the same observation several times in a row, being able to count the number of steps that have been taken by the system.

If we want to extend this work to timed systems, we need to add some more constraints. In particular, the strategy of the controller will have to be *stuttering invariant*, i.e. the strategy cannot be affected by a sequence of environment or time steps unless changes in the observations occur. In this sense the strategy is "triggered" by the changes of observations.

To illustrate the concepts of imperfect information and stuttering invariance consider the timed game automaton in Figure 5 modelling a production system for *painting* a box moving on a conveyor belt. The various locations indicate the position of the box in the system: in Sensor a sensor is assumed to reveal the presence of the box, in Sensed the box is moving along the belt towards the painting area, in Paint the actual painting of the box takes place, in Piston the box may be kick?'ed off the belt leading to Off; if the box is not kicked off it

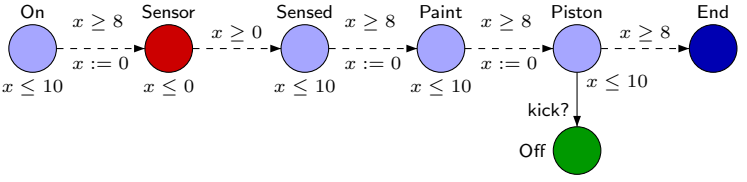


Fig. 5. Timed Game with Imperfect Information

ends in End. All phases are assumed to last between 8 and 10 seconds, except for the phase Sensor, which is instantaneous. The uncontrollability of this timing uncertainty is indicated by the dashed transitions between phases. The controller should now issue a *single* kick?'command at the appropriate moment in order to guarantee that the box will — regardless of the above timing uncertainty — be kicked off the belt. However the controller has *imperfect information* of the position of the box in the system. In particular, the controller cannot directly observe whether the box is in the Sensed, Paint or in the Piston phase nor can the value of the clock x be observed. Still equipping the controller with its own clock y – which it may reset and test (against a finite number of predicates) – it might be possible to synthesize a control strategy despite having only partial information: in fact it may be deduced that the box will definitely be in the Piston area within 20-24 seconds after being sensed. In contrast, an increased timing uncertainty where a phase may last between 6 and 10 seconds will make a single-kick? strategy impossible.

4.2 Observation-Based Stuttering Invariant Strategies

In the untimed setting of [16,29], each state change produces a new observation. For instance, if a run of the system is

$$\rho = l_0 \xrightarrow{a_0} l_1 \xrightarrow{a_1} l_2 \cdots \xrightarrow{a_n} l_{n+1}$$

we make the observation $\mathsf{Obs}(\rho) = \mathsf{Obs}(l_0)\mathsf{Obs}(l_1)\cdots\mathsf{Obs}(l_{n+1})$ where Obs is a mapping from states to a finite set of observations. In the previous example, even if Sensed, Paint and Piston produce the same observation, say 01, we could deduce where the box is by counting the number of 01.

Also, if each state change produces a new observation in the untimed setting, it cannot be a realistic assumption in the timed setting: time is continuous and the state of the system continuously changes.

A more realistic assumption about the system under observation is that the controller can only see *changes* of observations. In the previous piston example, assume the system makes the following steps:

$$(\mathsf{On}, x = 0) \xrightarrow{8} (\mathsf{On}, x = 8) \rightarrow (\mathsf{Sensor}, x = 0) \rightarrow (\mathsf{Sensed}, x = 0) \cdots$$
$$\cdots \xrightarrow{9} (\mathsf{Sensed}, x = 9) \rightarrow (\mathsf{Paint}, x = 0)$$

The sequence of observations the controller makes is: On Sensor 01 if $\mathsf{Obs}(\mathsf{On}) = \mathsf{On}$, $\mathsf{Obs}(\mathsf{Sensor}) = \mathsf{Sensor}$ and $\mathsf{Obs}(\mathsf{Sensed}) = \mathsf{Obs}(\mathsf{Paint}) = 01$.

This is why in [14] we consider *stuttering-free* observations. We assume we are given a finite set of observations $\mathcal{O} = \{o_1, o_2, \cdots, o_k\}$ and a mapping $\mathsf{Obs} : L \times \mathbb{R}^X_{\geq 0} \to \mathcal{O}$ (*i.e.* from the state space of the timed game automaton to $\mathcal{O}$). Given a run of timed game automaton

$$\rho = (l_0, v_0) \xrightarrow{e_0} (l_1, v_1) \xrightarrow{e_1} (l_2, v_2) \cdots \xrightarrow{e_n} (l_{n+1}, v_{n+1}) \cdots$$

we can define the observation of ρ by the sequence:

$$\mathsf{Obs}(\rho) = \mathsf{Obs}(l_0, v_0)\mathsf{Obs}(l_1, v_1) \cdots \mathsf{Obs}(l_{n+1}, v_{n+1}) \cdots$$

The stuttering-free observation of ρ is $\mathsf{Obs}^*(\rho)$ and is obtained from $\mathsf{Obs}(\rho)$ by collapsing successive identical observations $o_1 o_1 \cdots o_1$ into one o_1. In this setting the controller has to make a decision on what to do after a finite run ρ, according to the stuttering-free observation of ρ. Let f be a strategy in $\mathsf{Strat}(G)$. f is (observation based) *stuttering invariant* if for all ρ, ρ', finite runs in $\mathsf{Runs}(G)$, if $\mathsf{Obs}^*(\rho) = \mathsf{Obs}^*(\rho')$ then $f(\rho) = f(\rho')$.

The control problem under partial observation thus becomes: given a reachability game (G, K), is there an observation based stuttering invariant strategy to win (G, K)?

This raises an issue about the shapes of observations in timed systems: assume for the piston game, we define two observations $o_1 = (\mathsf{On}, x \leq 3)$ and $o_2 = (\mathsf{On}, x > 3)$. Given any finite run $(\mathsf{On}, x = 0) \xrightarrow{r} (\mathsf{On}, x = r)$ with $r \leq 10$ there is one stuttering-free observation: either o_1 if $r \leq 3$ or $o_1 o_2$ if $r > 3$. Still we would like our controller to be able to determine its strategy right after each change of observations, *i.e.* the controller continuously monitors the observations and can detect rising edges of each new observation. This implies that a *first instant* exists where a change of observations occurs. To ensure this we can impose syntactic constraints on the shape of the zones that define observations. They must be conjunctions of constraints of the form $k_1 \leq x < k_2$ where x is a clock and $k_1, k_2 \in \mathbb{N}$.

4.3 Playing with Stuttering Invariant Strategies

The controller has to play according to *(observation based) stuttering invariant strategies* (OBSI strategies for short). Initially and whenever the current observation of the system state changes, the controller either proposes a controllable action $c \in \mathsf{Act}_c$, or the special action λ (do nothing *i.e.* delay). When the controller proposes $c \in \mathsf{Act}_c$, this intuitively means that he wants to play the action c as soon as this action is enabled in the system. When he proposes λ, this means that he does not want to play any discrete actions until the next change of observation, he is simply waiting for the next observation. Thus, in the two cases, the controller sticks to his choice until the observation of the system changes: in this sense he is playing with an observation based stuttering invariant strategy. Once the controller has committed to a choice, the environment decides of the evolution of the system until the next observation. The game can be thought of to be a two-player game with the following rules:

1. if the choice of the controller is a discrete action $c \in \mathsf{Act}_c$, the environment can choose to play, as long as the observation does not change, either (i) discrete actions in $u \in \mathsf{Act}_u \cup \{c\}$ or (ii) let time elapse as long as c is not enabled. Thus it can produce sequences of discrete and time steps that respect the (i) and (ii) with action c being urgent,
2. if the choice of the controller is the special action λ the environment can choose to play, as long as the observation does not change, any of its discrete actions in Act_u or let time pass; it can produce sequences of discrete and time steps respecting the previous constraints;
3. the turn is back to the controller as soon as the next observation is reached. We have imposes special shape of constraints to ensure that a next first new observation always exists.

The previous scheme formalizes the intuition of observation based stuttering invariant strategies.

To solve the control problem for G under partial observation given by a finite set $\mathcal{O}$, we reduce it to a control problem on a new game G' under full observation.

4.4 An Efficient Algorithm for Partially Observable Timed Games

The reduction we have proposed in [14] follows the idea of *knowledge based subset construction* for discrete games introduced in [16,29].

Let $G = (L, \ell_0, \mathsf{Act}, X, E, \mathsf{Inv})$ be a TGA, and $\mathsf{Obs} : L \times \mathbb{R}^X_{\geq 0} \to \mathcal{O}$ be an observation map. Let K be particular location such that (G, K) is a reachability game and there is an observation o s.t. $\mathsf{Obs}(s) = o \iff s \in K$ *i.e.* the controller can observe if the system is in a winning state. We use $\mathsf{Obs}(K)$ for this particular observation. From G we build a finite discrete game $\tilde{G}$ the states of which are unions of pairs of symbolic states (l, Z) ($\ell \in L$ and Z is a zone). Moreover, we require that each state of $\tilde{G}$ contains pairs (l, Z) that have the same observation.

Each set of states S of $\tilde{G}$ corresponds to a set of points where, in the course of the game G, the controller can choose a new action because a new observation has just been seen. The controller can choose either to do a $c \in \mathsf{Act}_c$ or to let time pass (λ). Once the controller has made a choice, it cannot do anything until a new observation occurs. Given a state $(l, v) \in S$ and a choice a of the controller, we can define the tree $Tree((l, v), a)$ of possible runs starting in (l, v): this tree is just the unfolding of the game where we keep only the branches with actions in $\mathsf{Act}_u \cup \{a\}$ (see Figure 6, left). In this section, we assume that every finite run can be extended in an infinite run. Then on each infinite branch of this tree,

1. either there is a state with an observation o' different from $\mathsf{Obs}(l, v)$. In this case we can define the first (time-wise) state with a new observation. Such an example is depicted on Figure 6 (left): from (l, v), there is a first state with a new observation in o_1 and in o_2. Because we require that our observations have special shapes, this first state always exists.
2. or all the states have the same observation on the branch: this is depicted by the infinite path starting from (l', v').

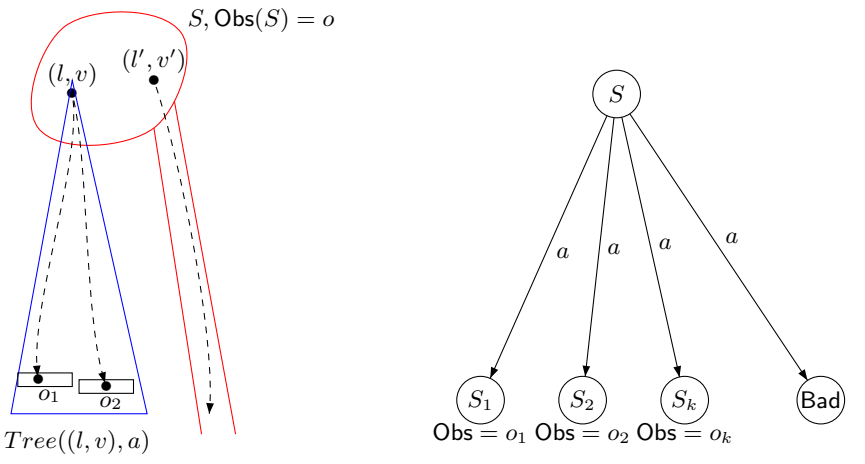


Fig. 6. From G to $\tilde{G}$

We denote $\mathsf{Next}_a(l, v)$ the set of first (time-wise) states with a new observation that can be reached from (l, v) if the controller plays a. If there is an infinite run from (l', v') on which all the states have the same observation, we say that (l', v') is a *sink* state for a.

We can now define the game $\tilde{G}$ as follows (Figure 6, right):

- the initial state of $\tilde{G}$ is $\{(\ell_0, \mathbf{0})\}$;
- let S be a state of $\tilde{G}$ with an observation different from $\mathsf{Obs}(K)$ (not winning). Let $a \in \mathsf{Act}_c \cup \{\lambda\}$. If there is a state (l', v') which is a sink state for a, we add a transition (S, a, Bad) in $\tilde{G}$.
- for each $o_i \in \mathcal{O}$ with $o_i \neq \mathsf{Obs}(S)$, if[3] $\mathsf{Next}_a(S) \cap o_i \neq \emptyset$ we add a transition (S, a, S_i) with $S_i = \mathsf{Next}_a(S) \cap o_i$, in $\tilde{G}$.

Given a state S of $\tilde{G}$, we let $\mathsf{Enabled}(S)$ be the set of actions a s.t. (S, a, S') for some S'. A state S of $\tilde{G}$ is winning if its observation is $\mathsf{Obs}(K)$. We let $\tilde{K}$ be the set of winning states of $\tilde{G}$. Notice that $\tilde{G}$ contains only controllable actions. Still $\tilde{G}$ is non-deterministic and thus it can be considered to be a two-player game: from S, the controller chooses an action and the environment chooses the next state among the successors of S by a. This construction of $\tilde{G}$ has the following property:

Theorem 3 ([14]). *The controller has a observation-based stuttering invariant strategy in (G, K) iff there is a winning strategy in $(\tilde{G}, \tilde{K})$.*

We can prove that $(\tilde{G}, \tilde{K})$ is finite and furthermore solving (G, K) amounts to solving a finite non-deterministic game. As we have an efficient algorithm, OTFUR (Fig. 3) to solve this type of games we can solve (G, K). To obtain an efficient algorithm written for (G, K) we simply have to use the Next operator to compute the transition relation "as needed" in OFTUR. Moreover, in case a state $(l, v) \in S$ is a

[3] We let $\mathsf{Next}_a(S) = \cup_{s \in S} \mathsf{Next}_a(s)$.

1: **Initialization**
2: $Passed \leftarrow \{\{s_0\}\}$ // *with* $s_0 = (l_0, \mathbf{0})$;
3: $ToExplore \leftarrow \{(\{s_0\}, \alpha, W') \mid \alpha \in \mathsf{Act}_c \cup \{\lambda\},\ o \in \mathcal{O}, o \neq \mathsf{Obs}(s_0),\ W' = \mathsf{Next}_\alpha(\{s_0\}) \cap o \wedge W' \neq \emptyset\}$;
4: $ToBackPropagate \leftarrow \emptyset$;
5: $Win[\{s_0\}] \leftarrow (\{s_0\} \in \tilde{K}\ ?\ 1 : 0)$;
6: $Losing[\{s_0\}] \leftarrow (\{s_0\} \notin \tilde{K} \wedge (ToExplore = \emptyset \vee \forall \alpha \in \mathsf{Act}_c \cup \{\lambda\}, \mathsf{Sink}_\alpha(s_0) \neq \emptyset)\ ?\ 1 : 0)$;
7: $Depend[\{s_0\}] \leftarrow \emptyset$;
8: **Main**
9: **while** $((ToExplore \cup ToBackPropagate \neq \emptyset)) \wedge Win[\{s_0\}] \neq 1 \wedge Losing[\{s_0\}] \neq 1))$ **do**
10: // *pick a transition from ToExplore or ToBackPropagate*
11: $e = (W, \alpha, W') \leftarrow pop(ToExplore)$ or $pop(ToBackPropagate)$;
12: **if** $W' \notin Passed$ **then**
13: $Passed \leftarrow Passed \cup \{W'\}$;
14: $Depend[W'] \leftarrow \{(W, \alpha, W')\}$;
15: $Win[W'] \leftarrow (W' \in \tilde{K}\ ?\ 1 : 0)$;
16: $Losing[W'] \leftarrow (W' \notin \tilde{K} \wedge \mathsf{Sink}_\alpha(W') \neq \emptyset\ ?\ 1 : 0)$;
17: **if** $(Losing[W'] \neq 1)$ **then**
18: $NewTrans \leftarrow \{(W', \alpha, W'') \mid \alpha \in \Sigma,\ o \in \mathcal{O},\ W' = \mathsf{Next}_\alpha(W) \cap o \wedge W' \neq \emptyset\}$;
19: **if** $NewTrans = \emptyset \wedge Win[W'] = 0$ **then** $Losing[W'] \leftarrow 1$;
20: $ToExplore \leftarrow ToExplore \cup NewTrans$;
21: **if** $(Win[W'] \vee Losing[W'])$ **then**
22: $ToBackPropagate \leftarrow ToBackPropagate \cup \{e\}$;
23: **else**
24: $Win^* \leftarrow \bigvee_{c \in \mathsf{Enabled}(W)} \bigwedge_{W \xrightarrow{c} W''} Win[W'']$;
25: **if** Win^* **then**
26: $ToBackPropagate \leftarrow ToBackPropagate \cup Depend[W]$;
27: $Win[W] \leftarrow 1$;
28: $Losing^* \leftarrow \bigwedge_{c \in \mathsf{Enabled}(W)} \bigvee_{W \xrightarrow{c} W''} Losing[W'']$;
29: **if** $Losing^*$ **then**
30: $ToBackPropagate \leftarrow ToBackPropagate \cup Depend[W]$;
31: $Losing[W] \leftarrow 1$;
32: **if** $(Win[W'] = 0 \wedge Losing[W'] = 0)$ **then**
33: $Depend[W'] \leftarrow Depend[W'] \cup \{e\}$;
34: **endif**
35: **endwhile**

Fig. 7. OTFPOR: **O**n-**T**he-**F**ly Algorithm for **P**artially **O**bservable **R**eachability

sink state for a and $\mathsf{Obs}(l, v) = \mathsf{Obs}(S) \neq \mathsf{Obs}(K)$, there is a transition (S, a, Bad) in $\tilde{G}$. We assume that $\mathsf{Obs}(\mathsf{Bad}) = \mathsf{Bad}$ and this observation in not winning so that if the controller plays a from S, then he looses which is consistent as there is a infinite run in G which does not encounter any state with the goal observation. In the version of OFTUR for partially observable timed games it is even more important to propagate backwards the information on losing states like Bad.

The version of the efficient algorithm OTFUR (Fig. 3) for discrete game can be adapted to deal with partially observable games: we obtain a new algorithm OTFPOR given in Fig. 7. In this version, $\mathsf{Sink}_a(S) \neq \emptyset$ stands for "there exists some $(l, v) \in S$ s.t. (l, v) is a sink state for a".

Lines 13 to 22 consist in determining the status of the new symbolic state W' and push it into *ToExplore* or *ToBackPropagate*. Lines 24 and 28 are rather expensive as we have to compute all the symbolic successors of W to determine its new status.

5 Conclusion and Future Work

In [13] we have proposed an efficient for the analysis of reachability timed games. It has been implemented in the tool UPPAAL-TIGA [8]. Recently in [14] we have proposed a version of this algorithm for partially observable timed games which is currently being implemented.

There are various directions in which this work can be extended:

- for finite state games, we can generalize our results for reachability games to safety games or more general games with Büchi objectives for instance. This can lead to efficient on-the-fly algorithm for finite games. Also we would like to study particular versions of Büchi objectives, *e.g.* with one repeated location as we may obtain more efficient algorithms for this case;
- for timed games, we can write a dual algorithm for safety objectives. Even in this case this is not always satisfactory as the controller could win with a so-called *zeno* strategy *i.e.* a strategy with which he keeps the game in a good state by playing infinitely many discrete controllable actions in a finite amount of time [15]. It is thus of great importance to ensure that the controller can win in a fair way. This can be encoded by a control objective which is a strengthened by a Büchi objective: we add a particular clock which is reset when it hits the value 1, and the Büchi objective is to hit 1 infinitely often which ensures time divergence. This Büchi objective can be encoded by a single location being forced infinitely often. If we can design an efficient algorithm for a single repeated state in the finite state case, we may be able to obtain efficient algorithms for synthesizing non-zeno controllers for safety timed systems.

These new efficient algorithms are going to be implemented in the tool UPPAAL-TIGA and it is expected that new useful practical results will be obtained for real case-studies.

Acknowledgements

The author wishes to thank Didier Lime and Jean-François Raskin for their careful reading and useful comments on preliminary versions of this paper.

References

1. Abdeddaïm, Y., Asarin, E., Maler, O.: Scheduling with timed automata. Theor. Comput. Sci. 354(2), 272–300 (2006)
2. Altisen, K., Gossler, G., Pnueli, A., Sifakis, J., Tripakis, S., Yovine, S.: A framework for scheduler synthesis. In: IEEE Real-Time Systems Symposium, pp. 154–163 (1999)
3. Altisen, K., Tripakis, S.: Tools for controller synthesis of timed systems. In: Proc. 2nd Workshop on Real-Time Tools (RT-TOOLS'02), Proc. published as Technical Report 2002-025, Uppsala University, Sweden (2002)
4. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science (TCS) 126(2), 183–235 (1994)
5. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)
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. Elsevier Science, Amsterdam (1998)
8. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., Lime, D.: Uppaal-tiga: Time for playing games! In: Proceedings of 19th International Conference on Computer Aided Verification (CAV'07). LNCS, vol. 4590, pp. 121–125. Springer, Berlin, Germany (2007)
9. Bouyer, P., Cassez, F., Fleury, E., Larsen, K.: Optimal Strategies in Priced Timed Game Automata. BRICS Reports Series RS-04-0, BRICS, Denmark, Aalborg, Denmark, ISSN 0909-0878 (February 2004)
10. Bouyer, P., Chevalier, F.: On the control of timed and hybrid systems. EATCS Bulletin 89, 79–96 (2006)
11. Bouyer, P., D'Souza, D., Madhusudan, P., Petit, A.: Timed control with partial observability. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 180–192. Springer, Heidelberg (2003)
12. 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)
13. Cassez, F., David, A., Fleury, E., Larsen, K., 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)
14. Cassez, F., David, A., Larsen, K., Lime, D., Raskin, J.-F.: Timed Control with Observation Based and Stuttering Invariant Strategies. In: ATVA 2007. LNCS, vol. 4762, Springer, Heidelberg (2007)
15. 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)
16. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Algorithms for omega-regular games with imperfect information. In: CSL, pp. 287–302 (2006)
17. de Alfaro, L., Henzinger, T.A., Majumdar, R.: Symbolic algorithms for infinite-state games. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 536–550. Springer, Heidelberg (2001)

18. La Torre, S., Mukhopadhyay, S., Murano, A.: Optimal-reachability and control for acyclic weighted timed automata. In: Proc. 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002). IFIP Conference Proceedings, vol. 223, pp. 485–497. Kluwer Academic Publishers, Dordrecht (2002)
19. Laroussinie, F., Larsen, K.G.: CMC: A tool for compositional model-checking of real-time systems. In: Budkowski, S., Cavalli, A.R., Najm, E. (eds.) Proceedings of IFIP TC6 WG6.1 Joint Int. Conf. FORTE'XI and PSTV'XVIII, Paris, France. IFIP Conference Proceedings, vol. 135, pp. 439–456. Kluwer Academic Publishers, Dordrecht (1998)
20. Laroussinie, F., Larsen, K.G., Weise, C.: From timed automata to logic – and back. In: Hájek, P., Wiedermann, J. (eds.) MFCS 1995. LNCS, vol. 969, pp. 529–539. Springer, Heidelberg (1995)
21. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a Nutshell. Journal of Software Tools for Technology Transfer (STTT) 1(1-2), 134–152 (1997)
22. Liu, X., Smolka, S.: Simple Linear-Time Algorithm for Minimal Fixed Points. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 53–66. Springer, Heidelberg (1998)
23. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: Mayr, E.W., Puech, C. (eds.) STACS 95. LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)
24. Ramadge, P., Wonham, W.: The control of discrete event systems. Proc. of the IEEE 77(1), 81–98 (1989)
25. Rasmussen, J., Larsen, K.G., Subramani, K.: Resource-optimal scheduling using priced timed automata. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 220–235. Springer, Heidelberg (2004)
26. Thomas, W.: On the synthesis of strategies in infinite games. In: Mayr, E.W., Puech, C. (eds.) STACS 95. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995)
27. Tripakis, S., Altisen, K.: On-the-Fly Controller Synthesis for Discrete and Timed Systems. In: Wing, J.M., Woodcock, J.C.P., Davies, J. (eds.) FM 1999. LNCS, vol. 1708, pp. 233–252. Springer, Heidelberg (1999)
28. Wong-Toi, H.: The synthesis of controllers for linear hybrid automata. In: Proc. 36th IEEE Conference on Decision and Control, pp. 4607–4612. IEEE Computer Society Press, Los Alamitos (1997)
29. Wulf, M.D., Doyen, L., Raskin, J.-F.: A lattice theory for solving games of imperfect information. In: HSCC, pp. 153–168 (2006)

Undecidability of Universality for Timed Automata with Minimal Resources

Sara Adams, Joël Ouaknine, and James Worrell

Oxford University Computing Laboratory
{sara.adams,joel,jbw}@comlab.ox.ac.uk

Abstract. Timed automata were introduced by Alur and Dill in the early 1990s and have since become the most prominent modelling formalism for real-time systems. A fundamental limit to the algorithmic analysis of timed automata, however, results from the undecidability of the universality problem: does a given timed automaton accept every timed word? As a result, much research has focussed on attempting to circumvent this difficulty, often by restricting the class of automata under consideration, or by altering their semantics.

In this paper, we study the decidability of universality for classes of timed automata with minimal resources. More precisely, we consider restrictions on the number of states and clock constants, as well as the size of the event alphabet. Our main result is that universality remains undecidable for timed automata with a single state, over a single-event alphabet, and using no more than three distinct clock constants.

1 Introduction

Timed automata were introduced by Alur and Dill in [3] as a natural and versatile model for real-time systems. They have been widely studied ever since, both by practitioners and theoreticians. A celebrated result concerning timed automata, which originally appeared in [2] in a slightly different context, is the PSPACE decidability of the *language emptiness* (or *reachability*) problem.

Unfortunately, the *language inclusion* problem—given two timed automata $\mathcal{A}$ and $\mathcal{B}$, is every timed word accepted by $\mathcal{A}$ also accepted by $\mathcal{B}$?—is known to be undecidable. This severely restricts the algorithmic analysis of timed automata, both from a practical and theoretical perspective, as many interesting questions can be phrased in terms of language inclusion. Over the past decade, several researchers have therefore attempted to circumvent this negative result by investigating language inclusion, or closely related concepts, under various assumptions and restrictions. Among others, we note the use of (i) topological restrictions and digitization techniques: [10,6,18,15,17]; (ii) fuzzy semantics: [9,11,16,5]; (iii) determinisable subclasses of timed automata: [4,20]; (iv) timed simulation relations and homomorphisms: [21,13,12]; and (v) restrictions on the number of clocks: [19,7].

The undecidability of language inclusion, first established in [3], derives from the undecidability of an even more fundamental problem, that of *universality*:

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 25–37, 2007.

does a given timed automaton accept every timed word? Research has shown the undecidability of universality to be quite robust, although it does break down under certain (fairly stringent) hypotheses, which we survey in Section 3.

The goal of the present paper is to study the (un)decidability of universality for classes of timed automata with *minimal resources*. Here 'resource' refers to quantities such as number of discrete states, number of clocks, size of the alphabet, number or magnitude of clock constants, etc. Our main result is that the universality problem remains undecidable for timed automata with a single state, over a single-event alphabet, and using the clock constants 0 and 1 only.[1]

At a conceptual level, one can paraphrase this result as asserting the existence of an undecidable problem for *stateless* and *eventless* real-time systems. Any computation carried out by such a device must rely exclusively on clocks, which can be viewed as some kind of unwieldy 'analog' memory. Clocks can only be reset (to zero) and compared against the constants 0 and 1;[2] moreover, they continually increase with time, and hence are quite poor at holding information over any given period of time.

One potential application of our work lies in establishing further decision problems about real-time systems to be undecidable; in this respect, the absence of any state or event structure in the formulation of our problem suggests it should be a well-suited target for a wide variety of real-time formalisms.

This paper summarises the key ideas and constructions of [1], to which we refer the reader for full details.

2 Timed Automata

Let X be a finite set of clocks, denoted x, y, z, etc. We define the set Φ_X of clock constraints over X via the following grammar, where $k \in \mathbb{N}$ stands for any non-negative integer, and $\bowtie \in \{=, \neq, <, >, \leq, \geq\}$ is a comparison operator:

$$\phi ::= \mathbf{true} \mid x \bowtie k \mid \phi \wedge \phi \mid \phi \vee \phi.$$

A *timed automaton* is a six-tuple $(\Sigma, S, S_0, S_f, X, \Delta)$, where

- Σ is a finite set (alphabet) of events,
- S is a finite set of states,
- $S_0 \subseteq S$ is a set of start states,
- $S_f \subseteq S$ is a set of accepting states,
- X is a finite set of clocks, and
- $\Delta \subseteq S \times S \times \Sigma \times \Phi_X \times \mathcal{P}(X)$ is a finite set of transitions. A transition (s, s', a, ϕ, R) allows a jump from state s to s', consuming event $a \in \Sigma$ in the process, provided the constraint ϕ on clocks is met. Afterwards, the clocks in R are reset to zero, while all other clocks remain unchanged.

[1] This result holds over *weakly monotonic* time; over *strongly monotonic* time, we require the clock constants 1, 2, and 3 instead. Full details are presented in Section 3.

[2] Note that we do not allow comparing clocks against each other.

Remark 1. An important observation is that *diagonal* clock constraints (of the form $x - y \bowtie k$) are not allowed in our model of timed automata. This restriction strengthens our main single-state undecidability results, in that multiple states cannot simply be encoded through the fractional ordering of clock values; see [22].

We will abuse notation and allow transitions to be labelled by *sets* of events, in addition to individual events. A transition labelled by $U \subseteq \Sigma$ simply corresponds to $|U|$ copies of the transition, one for each event in U.

For the remainder of this section, we assume a fixed timed automaton $\mathcal{A} = (\Sigma, S, S_0, S_f, X, \Delta)$.

A *clock valuation* is a function $\nu : X \to \mathbb{R}^+$, where $\mathbb{R}^+$ stands for the non-negative real numbers. If $t \in \mathbb{R}^+$, we let $\nu + t$ be the clock valuation such that $(\nu + t)(x) = \nu(x) + t$ for all $x \in X$.

A *configuration* of $\mathcal{A}$ is a pair (s, ν), where $s \in S$ is a state and ν is a clock valuation.

An *accepting run* of $\mathcal{A}$ is a finite alternating sequence of configurations and delayed transitions $\pi = (s_0, \nu_0) \xrightarrow{d_1,\theta_1} (s_1, \nu_1) \xrightarrow{d_2,\theta_2} \ldots \xrightarrow{d_n,\theta_n} (s_n, \nu_n)$, where $d_i \in \mathbb{R}^+$ and $\theta_i = (s_{i-1}, s_i, a_i, \phi_i, R_i) \in \Delta$, subject to the following conditions:

1. $s_0 \in S_0$, and for all $x \in X$, $\nu_0(x) = 0$.
2. For all $0 \leq i \leq n-1$, $\nu_i + d_{i+1}$ satisfies ϕ_{i+1}.
3. For all $0 \leq i \leq n-1$, $\nu_{i+1}(x) = \nu_i(x) + d_{i+1}$ for all $x \in X \setminus R_{i+1}$, and $\nu_{i+1}(x) = 0$ for all $x \in R_{i+1}$.
4. $s_n \in S_f$.

Each d_i is interpreted as the time delay between the firing of transitions, and each configuration (s_i, ν_i), for $i \geq 1$, records the data immediately following transition θ_i. Abusing notation, we also write runs in the form $(s_0, \nu_0) \xrightarrow{d_1,a_1} (s_1, \nu_1) \xrightarrow{d_2,a_2} \ldots \xrightarrow{d_n,a_n} (s_n, \nu_n)$ to highlight the run's events.

A *timed word* is a pair (σ, τ), where $\sigma = \langle a_1 a_2 \ldots a_n \rangle \in \Sigma^*$ is a word and $\tau = \langle t_1 t_2 \ldots t_n \rangle \in (\mathbb{R}^+)^*$ is a non-decreasing sequence of real-valued timestamps of the same length.

Such a timed word is *accepted* by $\mathcal{A}$ if $\mathcal{A}$ has some accepting run of the form $\pi = (s_0, \nu_0) \xrightarrow{d_1,a_1} (s_1, \nu_1) \xrightarrow{d_2,a_2} \ldots \xrightarrow{d_n,a_n} (s_n, \nu_n)$ where, for each $1 \leq i \leq n$, $t_i = d_1 + d_2 + \ldots + d_i$.

Remark 2. Our timed semantics is *weakly monotonic*, in that it allows multiple events to occur 'simultaneously', or, more precisely, with null-duration delays between them. We will also consider an alternative semantics, termed *strongly monotonic*, which requires the timestamps in timed words to be strictly increasing. As it turns out, our main undecidability results remain essentially the same, although some of the constructions have to be altered in places.

3 The Universality Problem

Consider a class of computational machines that act as language acceptors, such as finite automata or pushdown automata. A particular machine is said to be

universal if it accepts all possible words (over the relevant alphabet). The *universality problem*, for the class of machines in question, consists in determining whether a given machine is universal or not.

The universality problem is a cornerstone of formal language theory, and has been extensively studied in a wide array of contexts. Moreover, it is the simplest instance of the language inclusion problem, since virtually all interesting computational classes will comprise universal machines.

In this paper, we are interested in universality for certain restricted classes of timed automata. This problem splits naturally into two main instances, according to the assumption made on the monotonicity of time. A timed automaton is said to be *universal over weakly monotonic time* if it accepts all timed words (over its alphabet), and is said to be *universal over strongly monotonic time* if it accepts all timed words in which the timestamps are strictly increasing.

3.1 Background

The most fundamental result concerning universality for timed automata is Alur and Dill's proof of undecidability in the general case (over both weakly and strongly monotonic time) [3]. An examination of that proof reveals that universality is in fact undecidable for the class of timed automata having at most two clocks.

Much more recently, it was discovered that universality is decidable for timed automata having at most one clock (irrespective of the monotonicity of time) [19], using results from the theory of well-structured transition systems [8]. Together with Alur and Dill's work, this completely classifies the decidability of universality as a function of the number of clocks allowed.

The paper [19] also proves that universality is decidable for timed automata that only make use of the clock constant 0 in clock constraints. On the other hand, [3] shows that allowing any additional clock constant leads to undecidability.

Henzinger et al. introduced the notion of digitization in [10]. Using this technique, it is possible to show that, over weakly monotonic time, universality is decidable for open[3] timed automata. Universality is however undecidable for closed timed automata (regardless of the monotonicity of time) as well as for open timed automata over strongly monotonic time [18].

3.2 Main Results

The primary focus of this paper is to study universality for timed automata in which the number of states, the size of the alphabet, and the number of different clock constants are simultaneously restricted. Our main results are as follows:

Theorem 1. *Over weakly monotonic time, the universality problem is undecidable for timed automata with a single state, a single-event alphabet, and using clock constants* 0 *and* 1 *only.*

[3] A timed automaton is open if it only uses open (strict) comparison operators in clock constraints, i.e., $\{<, >, \neq\}$.

Theorem 2. *Over strongly monotonic time, the universality problem is undecidable for timed automata with a single state, a single-event alphabet, and using clock constants* 1, 2, *and* 3 *only.*

Remark 3. The above restrictions (number of states, size of alphabet, and number of clock constants) seemed to us the most natural and important ones to consider. We note that there would be no point in restricting in addition the number of clocks, since the total number of instances of such timed automata would then essentially be finite, vacuously making any decision problem decidable.

We also note that bounding the number of transitions, even on its own, would likewise result in a finite number of 'distinguishable' timed automata, and would therefore also make decidability issues moot.

The next section is devoted to proving Theorem 1. A proof sketch of Theorem 2 is given in Section 5.

4 Universality over Weakly Monotonic Time

At a high level, Alur and Dill's original undecidability proof runs as follows. Take a two-counter machine $\mathcal{M}$ with a distinguished halting state, and produce an encoding of its runs as timed words. In this encoding, a step or instruction of $\mathcal{M}$ is carried out every time unit; the values of the counters are encoded as repeated, non-simultaneous events, by exploiting the density of the real numbers to accommodate arbitrarily large numbers. One then manufactures a timed automaton $\mathcal{A}$ that accepts all timed words that do not correspond to some encoding of a valid halting run of $\mathcal{M}$. In other words, $\mathcal{A}$ is universal iff $\mathcal{M}$ does not halt, which immediately entails the undecidability of universality.

The construction of $\mathcal{A}$ makes heavy use of nondeterminism. More precisely, $\mathcal{A}$ accepts the encodings of all 'runs' that are either invalid or non-halting. The latter is easy to detect: a run is non-halting if it doesn't end in the distinguished halting state of $\mathcal{M}$. Invalid runs, on the other hand, exhibit at least one local violation, e.g., the consecutive values of one of the counters are inconsistent at some stage, or an illegal jump was made from one state to another, or the timed word does not respect the prescribed format, etc. In each case, $\mathcal{A}$ uses nondeterminism to 'find' and expose the local violation, and accept the corresponding timed word.

Our task here is to reproduce this overall approach under the stringent limitations of Theorem 1. The main ingredients are as follows:

1. We show that one can construct $\mathcal{A}$ to be a linear safety timed automaton, i.e., an automaton whose only loops are self-loops and all of whose states are accepting.
2. Moreover, we can reduce the alphabet of $\mathcal{A}$ to a single letter by encoding different symbols as fixed numbers of 'simultaneous' events.

3. Since $\mathcal{A}$ is linear, it can only change control states a bounded number of times over any run. Assume that all clocks initially have value greater than or equal to 1 (which can be achieved by accepting any behaviour during the first time unit). We can then reduce $\mathcal{A}$ to having a single state, by encoding other states through various combinations of certain clocks having value less than 1 and other clocks having value greater than or equal to 1.
4. Finally, the overall correctness of the construction requires that various technical conditions and invariants in addition be maintained throughout runs of $\mathcal{A}$; for example, if the delay between two consecutive events ever exceeds 1, then all subsequent behaviours should be accepted.

We now present the technical arguments in detail.

4.1 Two-Counter Machines

A *two-counter machine* $\mathcal{M}$ is a six-tuple (Q, q_0, q_f, C, D, Ξ), where Q is a finite set of states, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the halting state, C and D are two counters ranging over the non-negative integers, and Ξ is the transition relation. Both counters are initially empty, and $\mathcal{M}$ starts in state q_0. Every state except q_f has a unique outgoing transition in Ξ associated to it. Such a transition can either: (i) increment or decrement (if non-zero) one of the counters, and subsequently jump to a new state; or (ii) test one of the counters for emptiness and conditionally jump to a new state.

A *configuration* of $\mathcal{M}$ is a triple (q, c, d), where q is the current state and c, d are the respective values of the counters. A configuration is *halting* if it is of the form (q_f, c, d). We assume that the transition relation is fully deterministic, so that each non-halting configuration has a unique successor. It is well-known that the halting problem for two-counter machines, i.e., whether a halting configuration is reachable from $(q_0, 0, 0)$, is undecidable [14].

We associate to any given two-counter machine $\mathcal{M}$ a set of strongly monotonic timed words $L(\mathcal{M})$ that are encodings of the halting computations of $\mathcal{M}$. Our alphabet is $\Sigma = Q \cup \{a, b\}$ (where we assume $a, b \notin Q$). If $\mathcal{M}$ does not halt, then naturally we let $L(\mathcal{M}) = \emptyset$. We otherwise note that, since $\mathcal{M}$ is deterministic, it has at most one valid halting computation, which we denote $\pi = \langle (s_1, c_1, d_1)(s_2, c_2, d_2), \ldots, (s_n, c_n, d_n) \rangle$. We then include in $L(\mathcal{M})$ all timed words (σ, τ) that satisfy the following:

1. $\sigma = \sigma_{\text{init}} \sigma' \sigma_{\text{end}}$, where $\sigma' = (s_1 a^{c_1} b a^{d_1})(s_2 a^{c_2} b a^{d_2}) \ldots (s_{n-1} a^{c_{n-1}} b a^{d_{n-1}}) s_n$, and $\sigma_{\text{init}}, \sigma_{\text{end}} \subseteq \Sigma^*$. In other words, σ can be decomposed as a prefix σ_{init}, in which anything is allowed, followed by σ', which is a fairly straightforward encoding of π, and capped by a suffix σ_{end}, in which anything is allowed once again.[4] The values of the counters in configurations correspond to the number of consecutive a's, with b acting as a delimiter between the encodings of the first and the second counter.

[4] As we will aim to capture the complement of $L(\mathcal{M})$ using a safety automaton—whose language is therefore prefix closed—it is necessary for $L(\mathcal{M})$ to be suffix closed.

2. τ is strongly monotonic.
3. $\tau = \tau_{\text{init}}\tau'\tau_{\text{end}}$, where τ_{init}, τ', and τ_{end} are sequences whose lengths respectively match those of σ_{init}, σ', and σ_{end}. Moreover, all timestamps in τ_{init} are strictly less than 1.
4. The associated timestamp of each s_i in σ' is i. Moreover, if the timestamp of the first b in σ' is $1 + t_b$, then the timestamp of the ith b is $i + t_b$. An immediate consequence is that the time delay between successive events in σ' is always strictly less than 1.
5. For all $1 \leq i \leq n-2$: (i) if $c_{i+1} = c_i$, then for each a with timestamp t in the time interval $(i, i + t_b)$, there is an a with timestamp $t + 1$ in the time interval $(i+1, i+1+t_b)$; (ii) if $c_{i+1} = c_i + 1$, then for each a with timestamp $t + 1$ in the interval $(i + 1, i + 1 + t_b)$, except the last one, there is an a with timestamp t in the interval $(i, i+t_b)$; (iii) if $c_{i+1} = c_i - 1$, then for each a with timestamp t in the interval $(i, i + t_b)$, except the last one, there is an a with timestamp $t + 1$ in the interval $(i + 1, i + 1 + t_b)$; (iv) similar requirements hold for the second counter.

By construction, $\mathcal{M}$ halts iff $L(\mathcal{M})$ is not empty.

4.2 Linear Safety Timed Automata

Given a two-counter machine $\mathcal{M} = (Q, q_0, q_f, C, D, \Xi)$ as above, we sketch how to construct a timed automaton $\mathcal{A}$ that accepts precisely all strongly monotonic timed words that do not belong to $L(\mathcal{M})$. We ensure that $\mathcal{A}$ enjoys a number of technical properties, as follows:

1. $\mathcal{A}$ is *linear*, i.e., the only loops in its transition relation are self-loops. As a result, $\mathcal{A}$ can only change control states a bounded number of times over any run.
2. $\mathcal{A}$ is a *safety* timed automaton, i.e., all of its states are accepting.
3. As per the definition of $L(\mathcal{M})$, $\mathcal{A}$'s alphabet is $\Sigma = Q \cup \{a, b\}$.
4. $\mathcal{A}$ has a unique state, which we call the *sink* state, from which there is no transition to a different state. $\mathcal{A}$ moves into the sink state as soon as a violation is detected; we therefore postulate a Σ-labelled self-loop on the sink state, i.e., it accepts any behaviour.
5. $\mathcal{A}$ makes use of several clocks, some of which ensure that $\mathcal{A}$ only accepts strongly monotonic timed words.[5]
   Two important clocks are *abs* and *ev*. The clock *abs* is never reset and therefore indicates the total amount of time elapsed since the beginning of a run. After time 1, the clock *ev* is meant to be reset whenever an event occurs, unless a violation has already been detected. More precisely, *ev* obeys the following rules:
   - *ev* is never reset while $abs < 1$.

[5] Although the aim of the present section is to establish an undecidability result over *weakly* monotonic timed words, the automata we are considering here, which act as intermediate tools, are required to accept only *strongly* monotonic timed words.

- ev is not reset on transitions leading to the sink state; in particular, the sink state itself never resets ev.
- From every state, there is a Σ-labelled transition to the sink state guarded by the clock constraint $(abs > 1 \wedge ev \geq 1)$. In other words, once the absolute time exceeds 1, we require that events always occur less than 1 time unit apart, otherwise we record a violation.
- ev is always reset when $(abs > 1 \wedge ev < 1)$, and when $(abs = 1 \wedge ev = 1)$, unless the transition is headed to the sink state.

6. $\mathcal{A}$ accepts any behaviour before time 1; this is achieved by postulating a Σ-labelled self-loop on initial states, guarded by $abs < 1$.
7. $\mathcal{A}$ cannot leave an initial state until at least time 1; this is achieved by guarding all transitions leading away from an initial state by $abs \geq 1$.
8. Let us call an *inner* state one that is neither an initial state nor the sink state. If an inner state has a self-loop, it must be be labelled either by a or by $\Sigma \setminus \{q_f\}$.
9. After time 1, q_f can only label transitions that are headed into the sink state, i.e., once a violation has already been detected; otherwise, $\mathcal{A}$ would potentially accept encodings of valid halting computations of $\mathcal{M}$.

It remains to show that we can indeed construct a timed automaton $\mathcal{A}$ that captures the complement of $L(\mathcal{M})$ and has Properties 1–9. This is achieved as in Alur and Dill's proof, by amalgamating various simple automata, each of which captures some local violation of a halting computation of $\mathcal{M}$.[6] Suppose, for example, that from state q_3, $\mathcal{M}$ is supposed to increment counter D, and move to state q_5. One possible violation of this transition could consist in failing to increment D as per the encoding prescribed in Subsection 4.1, which requires the insertion of an extra a before q_5 in the corresponding timed word. Figure 1 depicts an automaton that captures precisely this behaviour in the case D is originally non-empty.

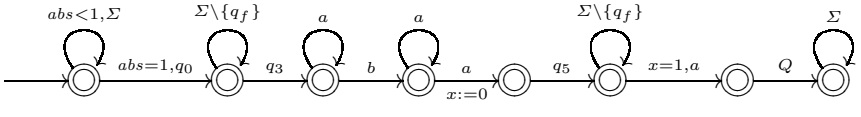


Fig. 1. A linear safety timed automaton that captures an incrementation violation on the second counter. In the interest of clarity, we have omitted data pertaining to Property 5 (strong monotonicity and the treatment of clock ev).

The (numerous) other cases are equally straightforward; full details can be found in [1].

4.3 Restricting to a Single Event

Let $\mathcal{M} = (Q, q_0, q_f, C, D, \Xi)$ be a two-counter machine. We give a way to encode, or 'flatten', any strongly monotonic timed language L over alphabet

[6] Note that our automata are trivially closed under union.

$\Sigma = Q \cup \{a, b\}$ into a weakly monotonic timed language $\widetilde{L}$ over the singleton alphabet $\{a\}$.

We define a renaming relation $\mathbf{R}$ from Σ to (untimed) words over alphabet $\{a\}$. This relation will be one-to-one for all events in Σ apart from q_f.

We first set $a\mathbf{R}a$ and $b\mathbf{R}aa$, i.e., the letter 'a' is renamed to the word 'a', and the letter 'b' is renamed to the word 'aa'. Next, let $s_1, \ldots, s_m$ be an enumeration of $Q \setminus \{q_f\}$. We set $s_i\mathbf{R}a^{i+2}$, for $1 \leq i \leq m$. Finally, for all $j \geq 1$, we set $q_f\mathbf{R}a^{m+2+j}$. In other words, q_f renames to the words a^{m+3}, a^{m+4}, a^{m+5}, ...

Note that the renaming relation is total and surjective, in that every event in Σ renames to one or more words over $\{a\}$, and every non-empty word over $\{a\}$ is the renaming of some event in Σ.

The renaming relation $\mathbf{R}$ lifts naturally to a renaming relation from strongly monotonic timed words over Σ to weakly monotonic timed words over $\{a\}$. Formally, write $u\mathbf{R}\widetilde{u}$ iff, for every event e with timestamp t in u, there are exactly k 'simultaneous' events a with timestamp t in $\widetilde{u}$, for some k such that $e\mathbf{R}a^k$, and vice-versa. For L a strongly monotonic timed language over Σ, we then let $\widetilde{L}$ be the image of L under this renaming. It is immediate that L is Σ-universal over strongly monotonic time iff $\widetilde{L}$ is $\{a\}$-universal over weakly monotonic time.

Let the timed automaton $\mathcal{A}$ be defined as in the previous subsection, i.e., the strongly monotonic language $L(\mathcal{A})$ of $\mathcal{A}$ is precisely the complement of $L(\mathcal{M})$ with respect to strongly monotonic timed words. Our next task is to define a single-event linear safety timed automaton $\widetilde{\mathcal{A}}$ that accepts precisely the language $\widetilde{L(\mathcal{A})}$.

$\widetilde{\mathcal{A}}$ can be obtained from $\mathcal{A}$ in a straightforward manner. Consider first an e-labelled transition of $\mathcal{A}$ that is not a self-loop, with $e \in \Sigma$. If $e \neq q_f$, then replace this transition by a sequence of k instantaneous a-labelled transitions, where $e\mathbf{R}a^k$. (Instantaneity can be enforced via a clock constraint such as $ev = 0$ on transitions.) If $e = q_f$, on the other hand, Property 9 guarantees that the transition is headed to the sink state, from which anything will be accepted, so it likewise suffices to replace the original transition by a sequence of $m + 3$ instantaneous a-labelled transitions that end in the sink state.

The case of self-loops is somewhat more subtle. It is clear that we need only handle self-loops on inner states, since any behaviour is allowed in any case in initial states (while $abs < 1$) and in the sink state. Property 8, however, guarantees that self-loops on inner states are either labelled by a or by $\Sigma \setminus \{q_f\}$. In the first case there is clearly nothing to change. In the second case, we assume that $\widetilde{\mathcal{A}}$ has the use of $m + 2$ special clocks $y_1, y_2, \ldots, y_{m+2}$. Whenever an event occurs, $\widetilde{\mathcal{A}}$ resets the y-clock of lowest index that is not already zero. If all y-clocks are zero, no transition is enabled. In this way, no more than $m + 2$ consecutive 'simultaneous' a's are possible from any inner state, in effect preventing encodings of q_f from occurring. (Note that as soon as some non-null amount of time elapses, all y-clocks automatically have non-zero values again.)

Stringing everything together, we have that $\mathcal{M}$ does not halt iff $\mathcal{A}$ is Σ-universal over strongly monotonic time iff the single-event timed automaton $\widetilde{\mathcal{A}}$ is $\{a\}$-universal over weakly monotonic time. Moreover, it is immediate from our construction that $\widetilde{\mathcal{A}}$ also satisfies Properties 1–9, *mutatis mutandis* (in particular,

$\widetilde{\mathcal{A}}$'s alphabet is simply $\{a\}$, and $\widetilde{\mathcal{A}}$ accepts both weakly and strongly monotonic timed words).

4.4 Restricting to a Single State

The final step is to transform $\widetilde{\mathcal{A}}$ into a single-state automaton $\widehat{\mathcal{A}}$ that accepts precisely the same language.

We have already observed that the linearity of $\widetilde{\mathcal{A}}$ places an upper bound on the total number of possible changes between states in any run of the automaton. Equivalently, the transition graph of $\widetilde{\mathcal{A}}$, with all self-loops removed, is simply a directed acyclic graph, or DAG. It follows that one can enumerate the states of $\widetilde{\mathcal{A}}$ as $s_1, s_2, \ldots, s_p$ so that the transition relation is monotone with respect to this enumeration: any transition $s_i \longrightarrow s_j$ of $\widetilde{\mathcal{A}}$ is such that $i \leq j$.[7] Note that this entails that s_p is the sink state.

Let $z_1, z_2, \ldots, z_p$ be p new clocks for $\widehat{\mathcal{A}}$ to use. Our rough intention is to encode state s_k by having, for every $1 \leq i \leq p$, $z_i < 1$ iff $i \leq k$. In order to adequately circumvent various technical difficulties (for instance, in initial states all clocks start with value 0), we refine this correspondence as follows:

1. Initial states are associated with the clock constraint:

$$\phi_{\text{init}} \equiv abs < 1 \vee (abs = 1 \wedge z_1 = 1 \wedge z_2 = 1 \wedge \ldots \wedge z_p = 1).$$

2. The sink state s_p is associated with the clock constraint:

$$\phi_{\text{sink}} \equiv (abs > 1 \wedge ev \geq 1) \vee (abs \geq 1 \wedge z_1 < 1 \wedge z_2 < 1 \wedge \ldots \wedge z_p < 1).$$

3. While $abs < 1$, we do not reset the z-clocks. Afterwards, on any transition, we systematically reset all z-clocks whose values are already strictly less than 1.
4. When entering any inner state s_k, we ensure, for all $1 \leq i \leq p$, that $z_i < 1$ iff $i \leq k$ (which is achieved by resetting all clocks z_i with $i \leq k$ on every transition with target s_k). It is clear that this discipline can be maintained, thanks to the monotonicity of the transition relation with respect to the enumeration of the states, and the fact that all z-clocks, upon leaving an initial state, have value at least 1.
   The clock constraint associated with an inner state s_k is therefore:

$$\phi_{s_k} \equiv abs \geq 1 \wedge \bigwedge\{z_i < 1 \,|\, i \leq k\} \wedge \bigwedge\{z_i \geq 1 \,|\, i > k\} \wedge \neg\phi_{\text{sink}}.$$

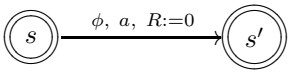
Recall that, after the initialisation phase (i.e., after time 1), consecutive events normally always occur strictly less than 1 time unit apart, otherwise the automaton transitions to the sink state (cf. Property 5 in Subsection 4.2). It therefore follows that, for any inner state s_k, the clock constraint ϕ_{s_k}, which holds upon

[7] Formally, such an enumeration can be obtained by topologically sorting the underlying transition DAG.

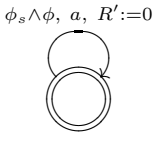
entering s_k, in fact holds continuously until the next transition occurs, unless that transition is headed for the sink state.

A second important observation is that, once ϕ_{sink} holds, it holds forever. Indeed, the clock ev is never reset in the sink state, so the clock constraint $ev \geq 1$, once true, never changes. On the other hand, if $ev < 1$, then we must have $z_i < 1$ for all i. Moreover, we also have $z_i \leq ev$ for all i, since the z-clocks all had to be reset when the sink state was first entered. It immediately follows that if the z_i's subsequently ever reach 1, then so does ev, so that ϕ_{sink} does indeed hold continuously.

The construction of $\widehat{\mathcal{A}}$ from $\widetilde{\mathcal{A}}$ is now straightforward. All states are merged into one, and every transition



of $\widetilde{\mathcal{A}}$ is replaced by a self-loop



in $\widehat{\mathcal{A}}$, where ϕ_s is the clock constraint associated with state s, and the new reset set R' comprises R together with all clocks z_i that are required to be reset upon entering s'.

It is easy to see that $\widehat{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$ simulate each other's behaviour almost perfectly. The only difference is that $\widehat{\mathcal{A}}$ can 'silently' transition from an initial or an inner state into the sink state, simply through the passage of time (this occurs if clock ev reaches 1). But in such a case, $\widetilde{\mathcal{A}}$'s next transition would take it to the sink state as well, so that there would be no visible difference in terms of behaviour (and language accepted) between the two automata.

We conclude that $\widehat{\mathcal{A}}$ is universal over weakly monotonic time iff the two-counter machine $\mathcal{M}$ does not halt. $\widehat{\mathcal{A}}$ has a single state and a singleton alphabet; moreover, it is straightforward to verify that the only clock constants required in its construction are 0 and 1.

This completes the proof of Theorem 1.

5 Universality over Strongly Monotonic Time

The overall approach to proving the undecidability of universality for single-state, single-event timed automata over strongly monotonic time is very similar to that for the weakly monotonic case. The main difference is that we cannot encode a plurality of events through instantaneous repetitions of a single event; in particular, we have no good way of mimicking the delimiter b. We must therefore alter our encoding of the halting computations of two-counter machines.

Let $\mathcal{M} = (Q, q_0, q_f, C, D, \Xi)$ be a two-counter machine. Recall that in Section 4, configurations of $\mathcal{M}$ were encoded over a unit-duration time interval. The essential difference is that here we encode configurations over three time units, as follows.

Let $s_1, \ldots, s_m$ be an enumeration of $Q \setminus \{q_f\}$, and consider a configuration (s, c, d) of $\mathcal{M}$. Given an integer k, we can encode this configuration over the time interval $[k, k+3)$ using a single event a. More precisely:

1. We encode the state s in the interval $[k, k+1)$, using i occurrences of a if $s = s_i$, and using $m+j$ occurrences of a, for any $j \geq 1$, if $s = q_f$. In addition, we require in both cases that the first a occur at time k.
2. We encode the value c of the first counter in the interval $(k+1, k+2)$ using c occurrences of a.
3. We encode the value d of the second counter in the interval $(k+2, k+3)$ using d occurrences of a.

The construction of a timed language $L'(\mathcal{M})$ containing encodings of the halting computation of $\mathcal{M}$ as strongly monotonic timed words can now proceed along the same lines as before. There is an 'initialisation' phase during the first three time units, followed by an encoding of the computation as successive configurations, and finally an arbitrary suffix. The counter discipline is the same as before, ensuring integrity by copying the contents of the counters exactly three time units apart.

Finally, a single-state, single-event linear safety timed automaton $\widehat{A'}$ can be constructed along the same lines as before to recognise the strongly monotonic complement of $L'(\mathcal{M})$. While not in the sink state, this automaton resets a distinguished clock every three time units; using the constants 1, 2, and 3,[8] it can therefore correctly check timed words for violations, as judged against the encoding described above. We leave the details of the construction to the reader.

This completes the proof sketch of Theorem 2.

6 Concluding Remarks

In this paper, we have studied the undecidability of the universality problem for timed automata with minimal resources, and have obtained some fairly stringent bounds: universality remains undecidable for timed automata with a single state, over a single-event alphabet, and using no more than three distinct clock constants.

One natural question is whether we can further tighten the restriction on clock constants. For example, it is an open problem whether the single constant 1 would suffice, over either weakly or strongly monotonic time.

References

1. Adams, S.: On the Undecidability of Universality for Timed Automata with Minimal Resources. MSc Thesis, Oxford University (2006)
2. Alur, R., Courcoubetis, C., Dill, D.: Model-Checking for Real-Time Systems. In: Proceedings of LICS 90, pp. 414–425. IEEE Computer Society Press, Los Alamitos (1990)

[8] Note that, over strongly monotonic time, there is no use for the clock constant 0.

3. Alur, R., Dill, D.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)
4. Alur, R., Fix, L., Henzinger, T.A.: Event-Clock Automata: A Determinizable Class of Timed Automata. Theoretical Computer Science 211, 253–273 (1999)
5. Alur, R., La Torre, S., Madhusudan, P.: Perturbed Timed Automata. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 70–85. Springer, Heidelberg (2005)
6. Bošnački, D.: Digitization of Timed Automata. In: Proceedings of FMICS 99 (1999)
7. Emmi, M., Majumdar, R.: Decision Problems for the Verification of Real-Time Software. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 200–211. Springer, Heidelberg (2006)
8. Finkel, A., Schnoebelen, P.: Well-Structured Transition Systems Everywhere! Theoretical Computer Science 256(1–2), 63–92 (2001)
9. 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)
10. Henzinger, T.A., Manna, Z., Pnueli, A.: What Good Are Digital Clocks? In: Kuich, W. (ed.) Automata, Languages and Programming. LNCS, vol. 623, pp. 545–558. Springer, Heidelberg (1992)
11. 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)
12. Kaynar, D.K., Lynch, N., Segala, R., Vaandrager, F.: Timed I/O Automata: A mathematical Framework for Modeling and Analyzing Real-Time Systems. In: Proceedings of RTSS 03, IEEE Computer Society Press, Los Alamitos (2003)
13. Lynch, N.A., Attiya, H.: Using Mappings to Prove Timing Properties. Distributed Computing 6(2), 121–139 (1992)
14. Minsky, M.L.: Recursive Unsolvability of Post's Problem of "Tag" and Other Topics in the Theory of Turing Machines. Annals of Mathematics 74(3), 437–455 (1961)
15. Ouaknine, J.: Digitisation and Full Abstraction for Dense-Time Model Checking. In: Katoen, J.-P., Stevens, P. (eds.) ETAPS 2002 and TACAS 2002. LNCS, vol. 2280, pp. 37–51. Springer, Heidelberg (2002)
16. Ouaknine, J., Worrell, J.: Revisiting Digitization, Robustness, and Decidability for Timed Automata. In: Proceedings of LICS 03, pp. 198–207. IEEE Computer Society Press, Los Alamitos (2003)
17. Ouaknine, J., Worrell, J.: Timed CSP = Closed Timed ε-Automata. Nordic Journal of Computing 10, 99–133 (2003)
18. Ouaknine, J., Worrell, J.: Universality and Language Inclusion for Open and Closed Timed Automata. In: Maler, O., Pnueli, A. (eds.) HSCC 2003. LNCS, vol. 2623, pp. 375–388. Springer, Heidelberg (2003)
19. Ouaknine, J., Worrell, J.: On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap. In: Proceedings of LICS 04, pp. 54–63. IEEE Computer Society Press, Los Alamitos (2004)
20. Raskin, J.-F.: Logics, Automata and Classical Theories for Deciding Real Time. PhD thesis, University of Namur (1999)
21. Taşiran, S., Alur, R., Kurshan, R.P., Brayton, R.K.: Verifying Abstractions of Timed Systems. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 546–562. Springer, Heidelberg (1996)
22. Tripakis, S.: Folk Theorems on the Determinization and Minimization of Timed Automata. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 182–188. Springer, Heidelberg (2004)

On Timed Models of Gene Networks*

Grégory Batt, Ramzi Ben Salah, and Oded Maler

VERIMAG, 2, av. de Vignate, 38610 Gieres, France
Gregory.Batt@imag.fr,Ramzi.Salah@imag.fr,
Oded.Maler@imag.fr

Abstract. We present a systematic translation from timed models of genetic regulatory networks into products of timed automata to which one can apply verification tools in order learn about the possible qualitative behaviors of the network under a whole range of uncertain delay parameters. We have developed a tool chain starting from a high-level description of the network down to an exhaustive analysis of its behavior. We have demonstrated the potential applicability of this framework on several examples.

1 Introduction

The evolving domain of *Systems Biology* attempts to advance the quality of biological models to become closer to models of simpler hard sciences like Physics. Given the complexity of such models and the difficulty in obtaining experimental results for determining exact model parameters, it is no big surprise that the engineering disciplines in general, and various "semantics and verification" sub communities, are among those who try to sell their modeling and analysis methodologies to biologists. The current paper is no exception as it attempts to demonstrate the applicability of timed systems for modeling and analysis of genetic regulatory networks.

We are concerned with models that cover a subset of what is going on inside a single cell where the major actors are *genes* at certain levels of activation ("expression") and the *products*, typically proteins that they produce in a cell. The interaction between these entities is often modeled by biologists using various forms of *interaction diagrams* that indicate the mutual influences among these entities. At this level of abstraction one may associate continuous variables to genes (to indicate the level of gene expression) and to products (to indicate their concentration in the cell). Based on the corresponding interaction diagrams, one can, in principle, derive dynamical models that track the evolution of these quantities over time. At this level of description, the dynamical model will be essentially a system of nonlinear differential equations[1] derived from the corresponding chemical processes.

* This work was partially supported by the French-Israeli project *Computational Modeling of Incomplete Biological Regulatory Networks.*

[1] Or a hybrid automaton with nonlinear dynamics in each mode, to accommodate for discrete modeling of changes in gene expression.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 38–52, 2007.

A major problem with such models is that the parameters of the equations are very difficult to obtain experimentally and can be known only within very large uncertainty margins. Such systems can, in principle, be analyzed using novel hybrid systems verification techniques, but although such techniques have recently matured for linear systems [ADF+06], their adaptation to nonlinear systems is only in its infancy. An alternative well-known modeling approach is based on *discrete* models where genes can be either *on* or *off* and the values of the product concentrations are discretized into a finite number of levels. In the extreme case when products can be only "absent" and "present" one can obtain an abstract Boolean model with a finite automaton dynamics of the form we all love and appreciate. Historically a *synchronous* Boolean model was first proposed by Kauffman [K69] followed by an alternative *asynchronous* model, proposed by Thomas in a series of papers and an influential book [TD90]. The rationale for Thomas' asynchronous model is that if two processes are active in parallel, one producing product p_1 and another producing p_2, it is very unlikely that both of them will terminate simultaneously in the "next" time step. Termination here means that their product concentrations will cross their respective thresholds between *present* and *absent*. The asynchronous model allows the processes to complete in any order.

While these asynchronous networks are more faithful to reality, they ignore information which may be known about the relative *speeds* of the processes, information that can be exploited to restrict the set of possible qualitative behaviors of the automaton. And indeed, recently Siebert and Bockmayr [SB06] proposed to enrich Thomas' model with *timing information* and replace automata with timed automata. They have used the tool UPPAAL to analyze such a model of a small network. In a completely different context, Maler and Pnueli [MP95] gave a formal treatment of asynchronous networks of Boolean gates with *uncertain (bi-bounded) delays* and a systematic method for translating such networks into timed automata. This framework has been since then the basis of numerous efforts for verification and timing analysis of such circuits [BMT99, BJMY02, BBM03]. In this paper we adapt this framework for the timed modeling of genetic regulatory networks and provide a tool chain for translating such networks into products of timed automata and analyzing its behaviors. The main improvement over [SB06] is in the *systematic* translation from timed gene networks to timed automata. At this point we do not claim having discovered any new biological result but rather demonstrate that timed models of non trivial phenomena can indeed be analyzed by our existing timed automata tools.

The rest of the paper is organized as follows. In Section 2 we describe our modeling framework based on genes, products, Boolean functions and delay operators. In Section 3 we show how such models are transformed into timed automata and discuss some anomalies inherent in discrete and timed modeling of continuous processes. To make such models more faithful to reality, we extend the framework of [MP95] in Section 4 to any finite number of concentration levels. Some experimental results on several examples are reported in Section 5, followed by some discussion of ongoing and future work. We assume familiarity with timed automata and present them in a quite informal manner to facilitate their comprehension by potential users.

2 Boolean Delay Networks

Let $G = \{g_1, \ldots, g_n\}$ be a set of "genes" viewed as Boolean variables which can be either *on* or *off*. Let $P = \{p_1, \ldots, p_n\}$ be a set of "products" or "proteins" that we assume to be represented by Boolean variables as well, where $p_i = 0$ means that the corresponding product is found in low concentration and $p_i = 1$ indicates that its concentration is high (absent/present in discrete parlance). We assume here that each gene g_i is responsible for the production of one product type p_i. Intuitively when g_i is 1 it will tend to produce p_i so that the latter, if absent, will sooner or later become present. On the other hand if g_i is off, so will become p_i eventually due to degradation. In control terms the value of g_i can be seen as a *reference signal* for the desired value of p_i, a value that it will reach if the process is not disturbed.

The feedback in the system is based on changing the state of the genes according to the concentration levels of the products. Among the common types of interaction between genes and the proteins they produce we mention:

- *Self-inhibition*: the presence of p_i turns g_i off;
- *A cascade of activations*: when p_i becomes present it turns on some g_j.

More generally, we assume the value of each g_i to be a *Boolean function* f_i of $p_1, \ldots, p_n$. We assume such a change in gene activation to be *instantaneous* upon the change in (discretized) concentration, if the latter changes the value of the corresponding Boolean function. On the other hand, the production and degradation of products is modeled as taking some time between initiation and termination.

This difference in modeling is justified by the nature and time scales of the underlying chemical processes. Producing one molecule of the product is already a very complex process involving numerous reactions. Changing the state of p_i from absent to present may involve producing thousands of such molecules. Although some of this can be done in parallel (or more accurately, with *pipelining*) it still may take a lot of time. On the other hand, activating or inhibiting a gene is a relatively-simpler and faster process where some molecule binds to some site and enables or inhibits the production process.[2]

Our modeling approach is based on the premise that at every time instant t, the state of a gene g_i is determined by the values of the P-variables *at time* t via a Boolean function f_i. On the other hand, the influence of each p_i on g_i is not immediate and is best expressed via a *delay operator*, a function whose output follows the input after some delay. Figure 1 illustrates graphically the relationship between G and P using a *block-diagram* formalism. The f_i-boxes react immediately to a change in their inputs and alter g_i, and the latter influences p_i after some delay specified by the D_i-boxes. Looking closer, each delay operator D_i can be characterized by a table of the following

[2] The dependence of such a binding event upon the presence level of some p is more of a stochastic nature as the more molecules we have, more this is likely to happen. In fact, the "real" story is more complex as the activation of a gene is not a Boolean business either, and there are different levels of gene expression, each leading to different production rates and delays. These rates depend on the concentration of the products via stochastics, but all this is beyond the scope of the present paper.

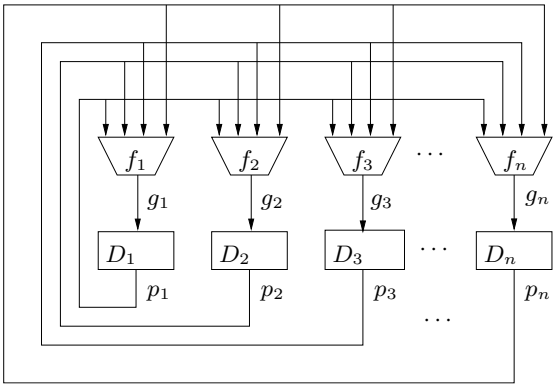


Fig. 1. A graphical representation of a gene network. The f_i's are arbitrary Boolean functions of P and the D_i's are the delay operators.

form:

p_i	g_i	p_i'	Δ
0	0	0	–
0	1	1	$[l^{\uparrow}, u^{\uparrow}]$
1	0	0	$[l^{\downarrow}, u^{\downarrow}]$
1	1	1	–

(1)

In this table, p_i' indicates the value of the product the systems "aims at" and should eventually reach. When p_i and g_i agree (00 or 11) the system is in a *stable* state and $p_i' = p_i$. In state 01 the product starts being produced and will become 1 if undisturbed. The time delay for moving from absence to presence is expressed using the interval $I^{\uparrow} = [l^{\uparrow}, u^{\uparrow}]$ to be explained in the sequel. Likewise, at state 10 the product degrades and will become 0 within $I^{\downarrow} = [l^{\downarrow}, u^{\downarrow}]$ time. To better explain the delay operator let us start with a deterministic delay model. Let $d^{\uparrow} = l^{\uparrow} = u^{\uparrow}$ and $d^{\downarrow} = l^{\downarrow} = u^{\downarrow}$. A typical behavior of this operator is illustrated in Figure 2-(a) where gene g_i is turned on at time t and then p_i follows and becomes present at time $t + d^{\uparrow}$. Later, at t', gene g_i is turned off and p_i completes its degradation to low level at $t' + d^{\downarrow}$.

It is, however, unrealistic to assume *exact* delays given the inherent noise in biological systems and general experimental limitations. Even in the absence of those, deterministic delays should be excluded due to the fact that discrete state 0 represents a *range* of concrete concentrations and it is clear that from each of them it will take a different amount of time to cross the threshold to reach the domain of 1. This feature is covered by our non-deterministic delay operator which allows the response of p_i to occur anywhere in the interval $[t + l, t + u]$, see Figure 2-(b).

We use Boolean signals[3] to formalize the D_i operators. A *Boolean signal* x is a function from the time domain $\mathbb{R}_+$ to $\mathbb{B} = \{0, 1\}$ which admits a partition of $\mathbb{R}_+$ into

[3] To avoid confusion with other meanings of "signals" in Biology, we stress that the word *signal* is used here in the sense used in of signal processing, that is, a function from time to some domain, a waveform, a temporal pattern.

a countable set of (left-closed right-open) intervals $I_0, I_1, \ldots$ where each I_i is of the form $[t_i, t_{i+1})$ such that if t and t' belong to the same interval, $x(t) = x(t')$. We will assume here, just for ease of notation, that all input signals start with 0, hence x is 0 in all intervals I_j with even j, and 1 when j is odd. The set $J(x) = t_1, t_2, \ldots$ is called the *jump set* of x, that is, the set of time points where the value of the signal changes.

Definition 1 (Delay Operator). *Let x and y be two Boolean signals with $J(x) = \{t_1, t_2, \ldots\}$ and $J(y) = \{s_1, s_2, \ldots\}$, and let D_i be a delay operator characterized by the delay parameters $l_i^{\uparrow}$, $u_i^{\uparrow}$, $l_i^{\downarrow}$ and $u_i^{\downarrow}$. We say that $y \in D_i(x)$ if for every j*

$$\begin{array}{ll} s_j \in [t_j + l_i^{\uparrow}, t_j + u_i^{\uparrow}] & \text{if } j \text{ is odd} \\ s_j \in [t_j + l_i^{\downarrow}, t_j + u_i^{\downarrow}] & \text{if } j \text{ is even} \end{array}$$

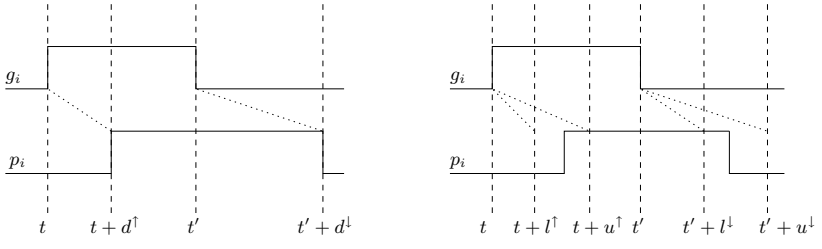


Fig. 2. (a) Deterministic delay $p_i = D_i(g_i)$; (b) Non-deterministic delay $p_i \in D_i(g_i)$

We can now define the semantics of the network (1), that is all the temporal behaviors it may generate over the values of the G and P variables. These are all the signals satisfying the following system of signal inclusions[4] for $i = 1, \ldots, n$:

$$\begin{array}{l} g_i = f_i(p_1, \ldots, p_n) \\ p_i \in D_i(g_i) \end{array} \tag{2}$$

3 Modeling with Timed Automata

Our translation from delay equations to timed automata is compositional as we build a timed automaton for each equation and inclusion in (2) so that the composition of these automata generates exactly the set of signals satisfying the equations. For each instantaneous relation $g_i = f_i(p_1, \ldots, p_n)$ we construct a one-state automaton over $(n+1)$-dimensional signals whose self-loop transitions are labeled by tuples $(g_i, p_1, \ldots, p_n)$ that satisfy the equation. The heart of our modeling approach is the automaton for the delay operator which can be seen as the continuous-time analog of the one-bit shift register.

The timed automaton of Figure 3-(a) realizes the delay operator $p \in D(g)$. We annotate states by the values of g and p where $\overline{p}$ stands for $p = 0$ and p stands for

[4] If the delay was deterministic, the second line would be replaced by $p_i = D_i(g_i)$ and there would be a unique solution from any given initial state.

$p = 1$. At state $\overline{pg}$ the product is absent, the gene is off and the automaton may stay in this stable state forever as long as g, which is viewed as an *input* for this automaton, remains off. When g is turned on, the clock c is reset to zero and the automaton moves to the excited state $g\overline{p}$ in which it can stay as long as $c < u^{\uparrow}$ but can leave to stable state gp, where the gene is on and the product is present, as soon as $c \geq l^{\uparrow}$.

The uncertainty in process duration is expressed in this automaton by the possibility to stay at an unstable state until $c = u$ but leave it as soon as $c \geq l$. An alternative modeling style is to move the non determinism to the triggering transition and making the stabilizing transition deterministic as in Figure 3-(b). Here instead of being reset to zero, the clock is set non deterministically to a value in $[0, u-l]$ and the transition guard is replaced with $c = u$. In both cases the result is the same, there are uncountably-many behaviors (and outputs) that the automaton may exhibit in the presence of one input. We will use the second approach in this paper as it extends more naturally from Boolean to multi-valued domains.

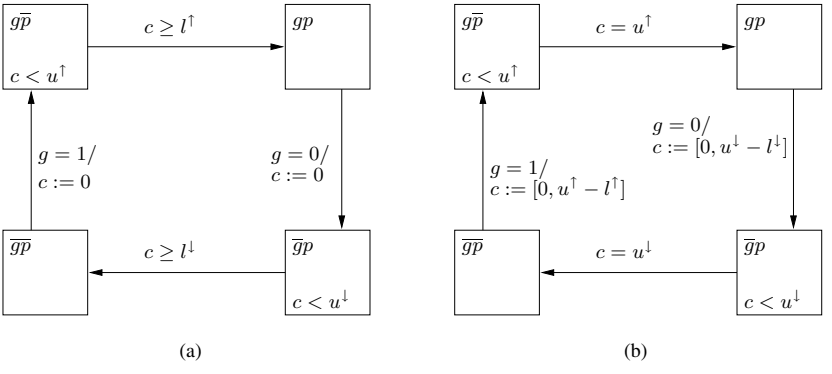


Fig. 3. Timed automaton models for the delay operator: (a) non determinism in the stabilizing transition; (b) non determinism in the exciting transition

Before proceeding further let us contemplate a bit on the relation between the delay bounds and the underlying continuous process. Figure 4 illustrates hypothetical production and decay processes to which such a timed discrete abstraction could correspond. To simplify the discussion assume that the concentration of p grows at a fixed rate $k^{\uparrow}$ when g is turned on, and decreases with rate $k^{\downarrow}$ when g is off.[5] The mapping from the concrete domain of concentrations, the interval $[0, 1]$ to $\{0, 1\}$ is based on partitioning the interval into $p^0 = [0, \theta]$ and $p^1 = [\theta, 1]$. The delay interval $D^{\uparrow}$ should thus indicate the minimal and maximal times it takes for a trajectory starting at *any* point in p^0 to cross the θ-threshold to p^1. Following this reasoning we obtain

$$D^{\uparrow} = [l^{\uparrow}, u^{\uparrow}] = [0, \theta/k^{\uparrow}] \quad \text{and} \quad D^{\downarrow} = [0, \theta/k^{\downarrow}].$$

The zero lower bounds come from the fact that the starting point can be a point in p^0 which is *as close as we want* to θ. Not having a positive lower bound is sometimes

[5] Approximate bounds can be derived for less trivial continuous dynamics using methods similar to those described in [HHW98, SKE00, F05].

considered an undesirable feature in timed models as it may create zero-time oscillations between 0 and 1, also known as Zeno behaviors. However, in our context, having a positive lower bound would exclude behaviors which are legitimate in the continuous context as we illustrate below.

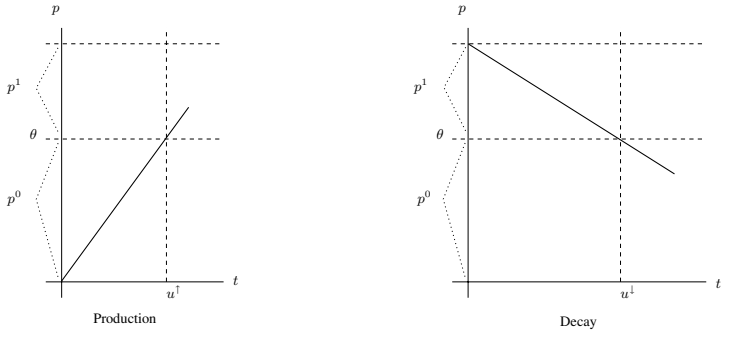


Fig. 4. An example of a continuous process which may underly the delay

Consider, for example, the negative feedback loop of Figure 5-(a) where the presence of p turns g off and its absence turns g on. Modeling the same phenomenon as a continuous system we will have a one-dimensional vector field like the one depicted in Figure 5-(b) which points toward the equilibrium θ from both sides. In the real noisy process, it is possible that the concentration will fluctuate around the equilibrium as in Figure 5-(c), but viewed discretely this is a Zeno behavior. A positive lower bound will prevent such behaviors and will force the system to stay, quite arbitrarily, on one side of θ for some time. On the other hand, the continuous "inverse image" of an oscillation between 0 and 1 includes unrealistic behaviors such as the one of Figure 5-(d) where the system exhibits large oscillations. Our modeling strategy is to use zero lower bounds but reduce their negative effects by moving from Boolean to multi-valued abstractions as will be described in Section 4. The automaton obtained for the negative feedback loop is shown in Figure 6-(a).

The alert reader might have noticed that we have not considered yet the case where the gene is turned off *before* the product becomes fully present. Such a situation may occur if the activation function of the gene depends on other products. In the automaton model this situation corresponds to g being turned on and then turned off at state $g\overline{p}$ before p becomes present (or the symmetric case when g is turned on at state $\overline{g}p$). This is modeled by the *regret* (or *abort*) transitions of the automaton of Figure 6-(b) that go from the excited state back to a stable state. Again, if we look at the continuous process, we cannot really know if it was aborted close to or far from the threshold, but since, using zero lower bounds, excitations are always accompanied by assignments of the form $c := [0, u]$, the timed model is conservative and covers all cases.

We construct such an automaton for each inclusion $p_i \in D_i(g_i)$ and a one-state automaton for each instantaneous relation $g_i = f_i(p_1, \ldots, p_n)$. Composing these automata we obtain a timed automaton that captures all the behaviors of the network [MP95]. We have implemented, within the IF tool suite [BGO+04], a translator from

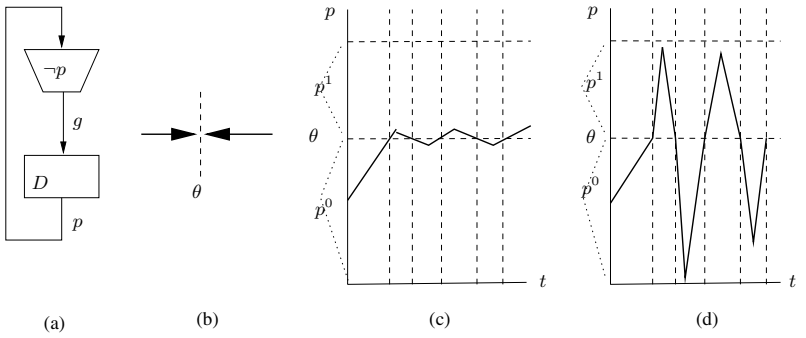


Fig. 5. (a) a negative feedback loop; (b) the corresponding one-dimensional continuous dynamical system; (c) a possible behavior of the underlying continuous dynamics; (d) a behavior which is impossible in the continuous system but is valid in the abstract timed model

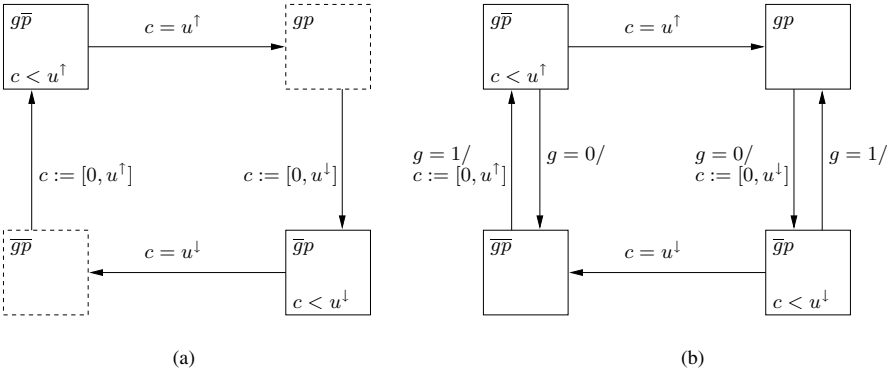


Fig. 6. (a) The automaton for the negative feedback network of Figure 5-(a). The automaton leaves states gp and $\overline{gp}$ immediately upon entrance; (b) A timed automaton model for the delay operator with zero lower bounds and the possibility of regret transitions from $g\overline{p}$ to $\overline{gp}$ and from $\overline{g}p$ to gp.

Boolean delay networks into timed automata which are then analyzed to produce the reachability graph which shows all the possible qualitative behaviors of the network when timing constraints are taken into account. These behaviors constitute a subset of what would be possible using an untimed model.

4 Multi-valued Models

The quality of the model can be improved significantly if we refine the discrete abstraction to admit *several levels* of concentration. To this end we replace the abstract set $\{0, 1\}$ by a finite set $\{0, 1, \ldots, m-1\}$, associated with a set of thresholds $0 < \theta_1 < \theta_2 < \ldots, < \theta_{m-1} < 1$ so that state i corresponds to the interval $p^i = [\theta_i, \theta_{i+1}]$. For each direction of evolution (production and degradation) we define an upper and lower

bound for moving between neighboring regions. To be more precise, let $v \xrightarrow{t} v'$ denote the fact that the underlying continuous process may go from v to v' in t time. Then the delay parameters are defined as

$$l_i^{\uparrow} = \min\{t : \theta_i \xrightarrow{t} \theta_{i+1}\} \qquad u_i^{\uparrow} = \max\{t : \theta_i \xrightarrow{t} \theta_{i+1}\}$$
$$l_i^{\downarrow} = \min\{t : \theta_i \xrightarrow{t} \theta_{i-1}\} \qquad u_i^{\downarrow} = \max\{t : \theta_i \xrightarrow{t} \theta_{i-1}\}$$

Whenever $g = 1$, the product level will move from p^i to p^{i+1} within the $[l_i^{\uparrow}, u_i^{\uparrow}]$ time interval and, likewise, it will move from p^i to p^{i-1} when $g = 0$. The extended delay operator is specified as follows:

g	p	p'	Δ	g	p	p'	Δ
0	0	0	$-$	1	0	1	$[l_0^{\uparrow}, u_0^{\uparrow}]$
0	1	0	$[l_1^{\downarrow}, u_1^{\downarrow}]$	1	1	2	$[l_1^{\uparrow}, u_1^{\uparrow}]$
0	2	1	$[l_2^{\downarrow}, u_2^{\downarrow}]$	1	2	3	$[l_2^{\uparrow}, u_2^{\uparrow}]$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
0	$m-1$	$m-2$	$[l_{m-1}^{\downarrow}, u_{m-1}^{\downarrow}]$	1	$m-1$	$m-1$	$-$

(3)

The corresponding automaton is shown in Figure 7. Its upper part corresponds to states where $g = 1$ and p is increasing, and the lower part to states where $g = 0$ and p is decreasing. The main attractive feature of this model is that it makes a distinction between entering a state via a *vertical* transition (which reverses the direction of evolution) and entering it via a horizontal transition (which is due to a monotone growth or decay process). In the latter case we *do not* need to use a zero lower bound because it is clear that the threshold was crossed *from below* (resp. above) and it will take some time between l_i and u_i to cross the next threshold in the same direction. Hence the transition from $(g, i-1)$ to (g, i) sets the clock to the interval $[0, u_i^{\uparrow} - l_i^{\uparrow}]$ while the transition from $(\overline{g}, i)$ to (g, i) sets the clock to the interval $[0, u_i^{\uparrow}]$ allowing an immediate transition from

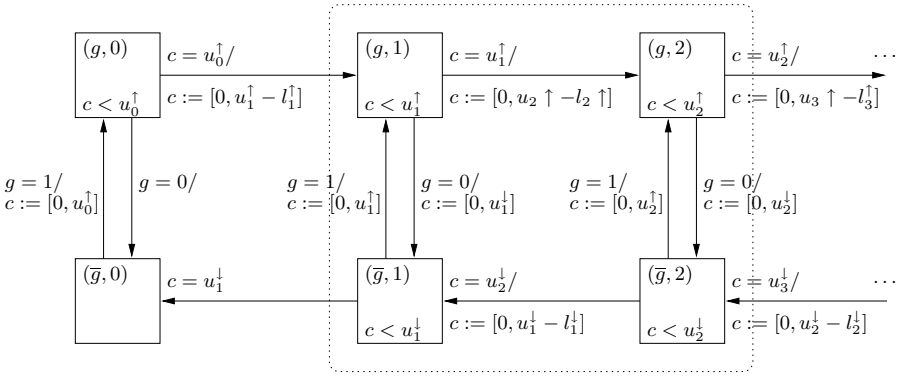


Fig. 7. A timed automaton model for the multi-valued delay operator. Zeno behaviors can now take place only among states that correspond to neighboring levels of concentration.

there to $(g, i+1)$. As a result, zero time cycles can now involve only states that correspond to *neighboring* levels of concentration, that is, $(g,i),(g,i+1),(\overline{g},i+1),(\overline{g},i)$. This way the deviation of the discrete timed model from the continuous one is reduced and tends to zero as m tends to infinity.

To complete the adaptation of the model to the multi-valued setting we replace each activation function of the form $f : \{0,1\}^n \to \{0,1\}$ by a function of the form $f' : \{0,\ldots,m-1\}^n \to \{0,1\}$ and have all the ingredients for translating the network in a timed automaton and analyzing its behaviors. Some experimental results are reported in the next section.

5 Experimental Results

We demonstrate the potential of our modeling framework and tools on several examples.

5.1 A Cross-Inhibition Network

We first consider a simple model of the lysis/lysogeny decision in the λ bacteriophage, a virus that infects bacteria. Following the infection, viruses can reproduce in bacteria in two different ways: *lysis* and *lysogeny*. In the first case, the virus multiplies in the bacterial cell and eventually kills the cell. In the second case, the genetic material of the virus integrates into the bacterial chromosome and replicates with it. A genetic regulatory network involving at least 5 viral genes is responsible for the choice between lysis and lysogeny. We study here a simple model of the core of the network that has been proposed as responsible for the lysis/lysogeny decision [TT95]. The model that we use is similar to the model used in [SB06] in their proposal to extend Thomas' model with timing information.

The network is represented by the diagram of Figure 8. It consists of two genes, *cI* and *cro*, that code for two repressor proteins, CI and Cro. More specifically, protein CI represses the expression of gene *cro*, whereas protein Cro represses the expression of gene *cI*, and at a higher concentration, the expression of its own gene. We denote the genes *cI* and *cro* by x and y and the proteins CI and Cro by X and Y.

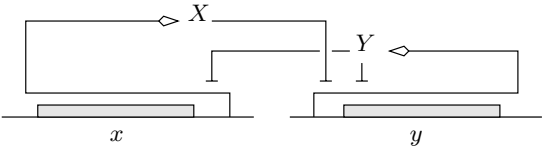


Fig. 8. A cross-inhibition network

In our formalism, we use Boolean variables g_x and g_y for the state of the genes, and two integer variables, $p_x \in \{0,1\}$ and $p_y \in \{0,1,2\}$ for the protein concentrations. The use of three concentration levels for protein Y is motivated by the fact that a moderate concentration of the protein is sufficient for the inhibition of gene x, whereas a high

concentration is needed to inhibit its own gene y. The following activation functions summarize this information:

p_x	p_y	g_x	g_y
0	0	1	1
0	1	0	1
0	2	0	0
1	0	1	0
1	1	0	0
1	2	0	0

The timed automata $\mathcal{A}_x$ and $\mathcal{A}_y$ for this example are depicted in Figure 9. As in [SB06], we have used the delay intervals $[5, 10]$, but as explained in Section 3, we assumed that some changes in protein concentration can happen instantaneously in order to guarantee that the timed model is a conservative over approximation of a continuous process.

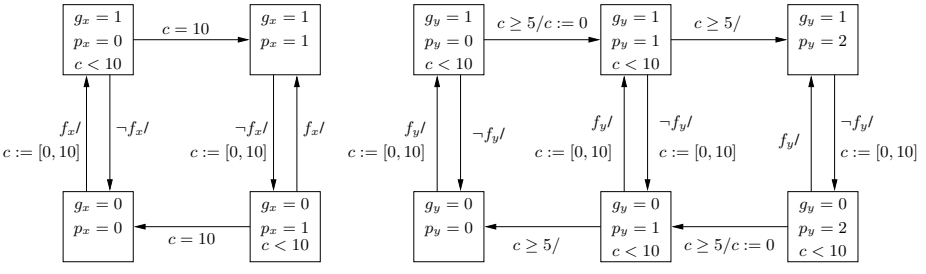


Fig. 9. The automata $\mathcal{A}_x$ and $\mathcal{A}_y$ for the cross-inhibition network

We analyzed this system using the IF toolbox [BGO+04], the successor of KRONOS [DOTY96]. The initial state of the system corresponds to the infection of a bacteria by a bacteriophage. In this state, both proteins are absent and both genes are *on*. The reachability graph generated by IF for this example is given in Figure 10-(a). Because of the interleaving semantics and the fact that changes in gene activity follow changes of protein concentration instantaneously, some states are left immediately upon entrance and have no biological significance. The toolbox collapses chains of states connected by immediate transitions into single states to obtain the automaton of Figure 10-(b).

A manual analysis of this reachability graph reveals that the system exhibits a mutual exclusion property, in the sense that it necessarily either ends up in a state where X is present and Y is absent (state 10), or oscillates between states where X is absent and Y is present in either medium (state 01) or high (state 02) concentration. The mutual exclusion property is a well-known property of cross-inhibition networks such as the one we study. Additional interesting timing properties can be inferred from the reachability graph. First, state 10 is the only state in which the system can remain forever and it can be reached from initial state 00 within time included in the interval $[0, 10]$. Secondly, it takes to the system between 5 and 20 time units to reach a state 02 having a high concentration of Y. Finally, the oscillations period is between 0 and 20 time units, that

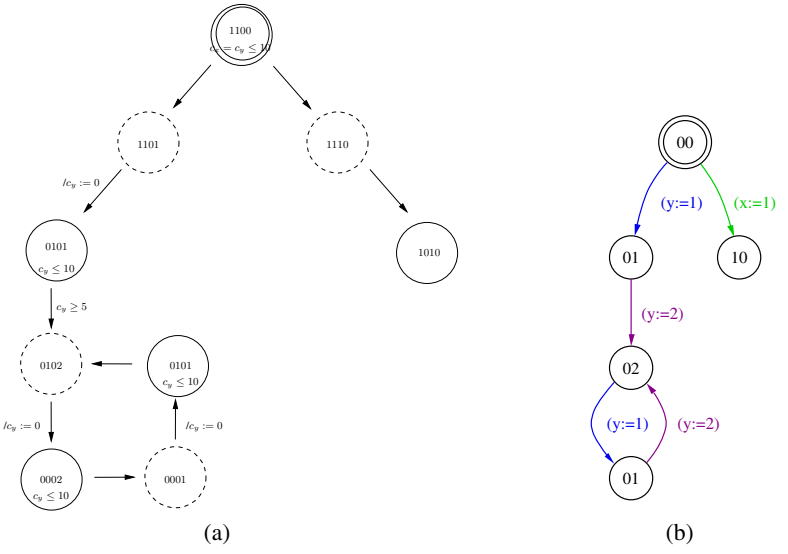


Fig. 10. (a) The reachability graph generated by IF for the cross-inhibition network. State are labeled by values of state variables (g_x, g_y, p_x, p_y). Two clocks c_x and c_y are used and the initial state is double circled. Dotted circles represent states that are left immediately. (b) The reachability graph after collapsing instantaneous transitions. State labels correspond to protein concentrations (p_x, p_y) and the transitions are labeled by changes in their values.

corresponds to all the cases between damped oscillations (period arbitrary close to 0) and sustained oscillations with maximum amplitude (period of 20 time units). It should be noted that the reachability graph makes a distinction between two instances of state 01, the first being reached from initial state 00 and hence having to wait at least 5 time units to move to 02, and the second reached from 02 and hence can return to it immediately. To summarize this example, we obtain the same qualitative results as in the untimed model (which suggest that this particular network is robust under delay variations), plus some additional timing information on the possible behaviors of the system.

5.2 Transcriptional Cascade in *E. coli*

As a second example, we study a transcriptional cascade of *E. coli* [HTW05] represented in Figure 11. It is made of four genes: *tetR*, *lacI*, *cI*, and *eyfp* that code respectively for three repressor proteins, TetR, LacI, and CI, and the fluorescent protein EYFP. The fluorescence of the system, due to the protein EYFP, can be measured. The system can be controlled by the addition or removal of a small diffusible molecule, aTc, in the growth media. More precisely, aTc binds to TetR and relieves the repression of *lacI*. One can check that the fluorescence of the system at steady state will be low for low aTc concentrations, and high for high aTc concentrations.

We have made a simple model of this system, assuming a maximal delay for protein production of 45 minutes for all proteins. Although these delays are biologically realistic they should not be considered as accurate and they are used only to demonstrate the

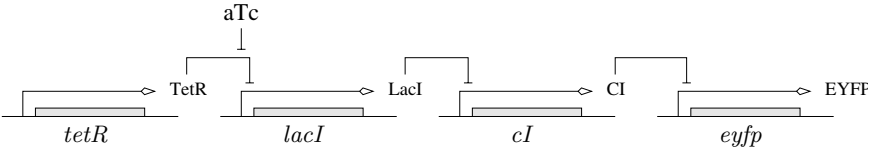


Fig. 11. A transcriptional cascade

capabilities of our tool. Using the IF toolbox, we computed the reachability graph for two different initial conditions, corresponding to high and low aTc concentrations. We obtained, respectively, graphs having 17 states and 34 transitions (see Figure 12), and 19 states and 38 transitions. The computations lasted less than one second on a standard PC. In both cases, these computations indicate that the system eventually stabilizes and always remains in a state in which the fluorescence level is consistent with what is experimentally observed: high fluorescence in presence of aTc, and low fluorescence in its absence. Manual analysis of these graphs revealed that the equilibrium state is necessarily reached in less that 6 hours. This upper bound is much larger than what has been observed experimentally [HTW05], and is due to the fact that we have used coarse Boolean abstractions of the concentration levels. Nevertheless, proving the existence of some upper bounds, which cannot be done using untimed models, is an important step toward using such a cascade as a module in a synthetic network.

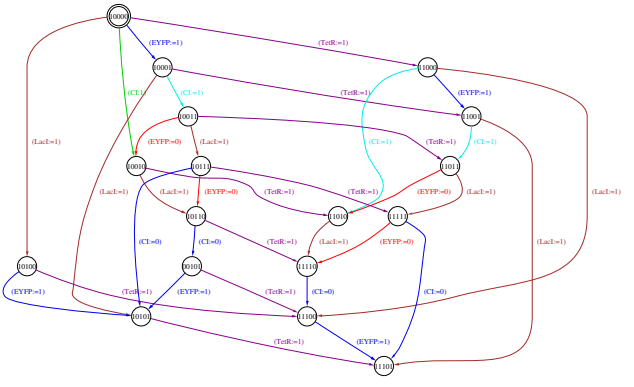


Fig. 12. The reachability graph for the transcriptional cascade in the *presence* of aTc. State labels correspond to $(p_{aTc}, p_{tetR}, p_{lacI}, p_{cI}, p_{eyfp})$. The initial state is 10000 (aTc present and EYFP absent) and the attractor state is 11101 (aTc present and EYFP present).

5.3 Nutritional Stress Response in *E. coli*

Our last example is based on a model of the nutritional stress response in *E. coli* which plays a crucial role in its survival. When confronted with a nutritional stress, this creature stops growing and enters in a dormant, resistant state. We use a simplified version of the elaborate model of this phenomenon proposed in [RJP+06]. The model consists of six genes, six proteins, and one additional variable encoding the presence or absence of nutrition. Since no timing information is available on this system we have arbitrarily used $[0, 45]$ minutes as a delay interval, just to check the feasibility of analysis.

The reachability graph, computed using the IF toolbox in less than one second, has 69 states and 209 transitions, and admits several cycles (Figure 13). However, the manual analysis of graphs of this size is less attractive and should be replaced with model checking against higher-level temporal properties that we intend to integrate into our toolbox. Nevertheless, this example shows that the exhaustive analysis of complex timed models of gene networks is feasible.

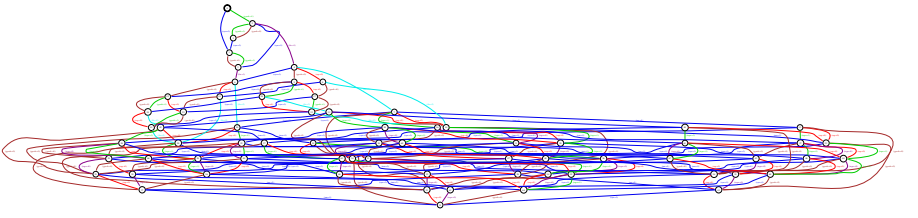


Fig. 13. The reachability graph for the nutritional stress response

6 Discussion

We have laid down some conceptual foundations for timed modeling of genetic regulatory network and other biological processes, provided tool support for the definition and analysis of such models and demonstrated its effectiveness on several examples. In particular, we have extended the framework of [MP95] from Boolean to multi-valued domains in a clean and systematic manner which reduces the effect of Zeno behaviors on the quality of the model. We have tested the computational effectiveness of our modeling approach on several non-trivial examples.

The modeling framework can be refined further to accommodate several levels of gene activation, each with different production rates and delays. Such an extension will require a careful examination of different ways to construct functions over bounded integer domains using a combination of logical and arithmetical operations.

Finally let us not delude ourselves that after having provided modeling and tool support, all that remains is to sit and wait for biologists to submit their models for analysis. Much is still to be done in promoting this class of models among them and in orienting their experiments to yield the information required to make these models useful. We have reasons to believe that this information could be, to a certain extent, easier to obtain than what is needed for meaningful continuous models, for example from micro-array experiments with a low sampling rate. If this is the case we can hope that timed models will find a significant niche in the discrete-to-continuous spectrum of dynamical system models used for biological purposes.

References

[ADF+06] Asarin, E., Dang, T., Frehse, G., Girard, A., Le Guernic, C., Maler, O.: Recent Progress in Continuous and Hybrid Reachability Analysis. In: CACSD (2006)

[BDL+01] Behrmann, G., David, A., Larsen, K.G., Möller, O., Pettersson, P., Yi, W.: UPPAAL - Present and future. In: CDC'01 (2001)

[BBM03] Ben Salah, R., Bozga, M., Maler, O.: On Timing Analysis of Combinational. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 204–219. Springer, Heidelberg (2004)

[BGO+04] Bozga, M., Graf, S., Ober, I., Ober, I., Sifakis, J.: The IF toolset. In: SFM (2004)

[BJMY02] Bozga, M., Jianmin, H., Maler, O., Yovine, S.: Verification of Asynchronous Circuits using Timed Automata. ENTCS 65 (2002)

[BMT99] Bozga, M., Maler, O., Tripakis, S.: Efficient Verification of Timed Automata using Dense and Discrete Time Semantics. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 125–141. Springer, Heidelberg (1999)

[DOTY96] Daws, C., Olivero, A., Tripakis, S., Yovine, S.: The tool KRONOS. In: Alur, R., Sontag, E.D., Henzinger, T.A. (eds.) Hybrid Systems III. LNCS, vol. 1066, pp. 208–219. Springer, Heidelberg (1996)

[F05] 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)

[HHW98] Henzinger, T.A, Ho, P.-H., Wong-Toi, H.: Algorithmic Analysis of Nonlinear Hybrid Systems. IEEE TRans. on Automatic Control 43, 540–554 (1998)

[HTW05] Hooshangi, S., Thiberge, S., Weiss, R.: Ultrasensitivity and Noise Propagation in a Synthetic Transcriptional Cascade. PNAS 102, 3581–3586 (2005)

[K69] Kauffman, S.A.: Metabolic Stability and Epigenesis in Randomly Constructed Genetic Nets. J. of Theoretical Biology 22, 437–467 (1969)

[MP95] Maler, O., Pnueli, A.: Timing Analysis of Asynchronous Circuits using Timed Automata. In: Camurati, P.E., Eveking, H. (eds.) CHARME 1995. LNCS, vol. 987, pp. 189–205. Springer, Heidelberg (1995)

[RJP+06] Ropers, D., de Jong, H., Page, M., Schneider, D., Geiselmann, J.: Qualitative Simulation of the Carbon Starvation Response in Escherichia coli. BioSystems 84, 124–152 (2006)

[SB06] Siebert, H., Bockmayr, A.: Incorporating Time Delays into the Logical Analysis of Gene Regulatory Networks. In: Priami, C. (ed.) CMSB 2006. LNCS (LNBI), vol. 4210, pp. 169–183. Springer, Heidelberg (2006)

[SKE00] Stursberg, O., Kowalewski, S., Engell, S.: On the Generation of Timed Discrete Approximations for Continuous Systems. Mathematical and Computer Modelling of Dynamical Systems 6, 51–70 (2000)

[TT95] Thieffry, D., Thomas, R.: Dynamical Behaviour of Biological Regulatory Networks: II. Immunity Control in Bacteriophage Lambda. Bulletin of Mathematical Biology 57, 277–295 (1995)

[TD90] Thomas, R., D'Ari, R.: Biological Feedback. CRC Press (1990)

Costs Are Expensive!

Patricia Bouyer[1,2,⋆] and Nicolas Markey[1]

[1] LSV, CNRS & ENS Cachan, France
[2] Oxford University Computing Laboratory, UK
{bouyer,markey}@lsv.ens-cachan.fr

Abstract. We study the model-checking problem for WMTL, a cost-extension of the linear-time timed temporal logic MTL, that is interpreted over weighted timed automata. We draw a complete picture of the decidability for that problem: it is decidable only for the class of one-clock weighted timed automata with a restricted stopwatch cost, and any slight extension of this model leads to undecidability. We finally give some consequences on the undecidability of linear hybrid automata.

1 Introduction

Since a couple of years, the verification technology for timed automata has evolved in several interesting directions, to answer new challenges posed by modern real-life systems, like the control of resource (e.g. energy) consumption. In that direction, weighted (or priced) timed automata [ALP01, BFH+01] have been designed as an extension of the timed automaton formalism, which uses observer variables to measure the performance of executions of the system. This model raises numerous interesting optimization problems. A number of them have been shown decidable, including optimal cost reachability [ALP01, BFH+01, BBBR07], optimal reachability in a multi-cost setting [LR05] or mean-cost optimal schedules [BBL04]. Note that the first and third problems even induce no extra complexity compared to the classical problems without optimization constraints (they are PSPACE-Complete).

Unfortunately, in general, adding resource consumption information is far from being free-of-charge! Indeed, to now, two main branches have been explored, which both lead either to negative results, or to complex algorithms. The first branch concerns the control problems, where a controller tries to minimize resource consumption, whatever an environment does: computing optimal cost is undecidable in general [BBR05], and this result holds even if the models have no more than three clocks [BBM06]. Similarly, the model checking of WCTL, a natural cost-extension of the branching-time logic CTL, has been investigated, and very similar undecidability results have been obtained [BBR04, BBM06], even when strong hypotheses are made on the cost [BBR06].

Recently, restriction of timed models to one clock has raised some interest in the community, with interesting complexity or decidability results, like

⋆ Partly supported by a Marie Curie fellowship, a program of the European Commission.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 53–68, 2007.

the NLOGSPACE-Completeness of reachability checking in one-clock timed automata [LMS04], or the decidability of emptiness checking in one-clock alternating timed automata over finite timed words [LW05, OW05, LW07, OW07]. In the context of weighted timed systems, this restriction also leads to nice improvements. Indeed, optimal timed games have been proven decidable under that restriction [BLMR06]. The same holds for the model checking of WCTL [BLM07], which even remains PSPACE (*i.e.*, has the same complexity as the model checking of TCTL, the classical timed extension of CTL, even under the single-clock restriction). However, not everything is decidable in this one-clock framework, and for instance, the model checking of WCTL* is still undecidable [BLM07].

In this paper, we tackle an obvious continuation of this literature, by considering the cost extension of the linear-time logic LTL, which we call WMTL (MTL, for "Metric Temporal Logic", is one of the classical timed extensions of LTL introduced by [Koy90], and WMTL can also be viewed as a weighted extension of MTL, hence the name). Indeed, it is known since [OW05] that the model-checking problem for MTL over timed automata accepting finite timed words is a decidable problem, whereas it becomes undecidable for infinite timed words [OW06]. We hence investigate the model checking problem for WMTL over weighted timed automata recognizing finite timed words, and draw a complete picture of the decidability results by proving that only the restriction to one-clock weighted timed automata using a single stopwatch-cost variable[1] is decidable, and that any single extension (like having a non-stopwatch cost variable, or two clocks, or two stopwatch cost variables) leads to undecidability. The decidability proof relies on technics developed in [OW05, OW07] (notice however that in our precise case, it is only valid for one-clock automata). The undecidability proofs rely on a reduction from the halting problem of two-counter machines and push ideas developed in [BBM06, BLM07] much further to get an undecidability result with only one clock in the model and a single cost variable. Finally, these undecidability results have some consequences on the undecidability landscape for linear hybrid automata.

2 Definitions

2.1 Weighted Timed Automata

In the whole paper, AP is a fixed, non-empty set of atomic propositions. In this section, we introduce the notion of *weighted timed automata* [ALP01] (also called *priced timed automata* [BFH$^+$01]), which is an extension of timed automata with a cost information (or weight) on both locations and edges. We first introduce usual notations and definitions for timed automata.

Given a finite set $\mathcal{X}$ of clocks, the set of clock valuations over $\mathcal{X}$ (that is, of applications $v\colon \mathcal{X} \to \mathbb{R}_{\geq 0}$) is denoted $\mathbb{R}_{\geq 0}^{\mathcal{X}}$. Given a valuation v and a nonnegative real $\tau \in \mathbb{R}_{\geq 0}$, the valuation $v + \tau$ is defined by $(v + \tau)(x) = v(x) + \tau$ for every $x \in \mathcal{X}$. The set $\mathcal{G}(\mathcal{X})$ denotes the set of *guards* over $\mathcal{X}$ which are finite

[1] A cost is stopwatch if it behaves like a clock that can be stopped and restarted.

conjunctions of atomic guards of the form $x \sim n$ where $x \in \mathcal{X}$ is a clock, $n \in \mathbb{N}$ is an integer constant, and $\sim$ is one of the symbols $\{<, \leq, =, \geq, >\}$. Notation $v \models g$ means that the valuation v satisfies the guard g (which is defined in the natural way). A *reset* $r \subseteq \mathcal{X}$ is a subset of $\mathcal{X}$ indicating which clocks are to be reset to 0; we write $v' = v[r \leftarrow 0]$ for the resulting valuation, *i.e.*, $v'(x) = 0$ if $x \in r$, and $v'(x) = v(x)$ otherwise.

Definition 1. *A* weighted timed automaton *(WTA in short) with k cost functions is a tuple $\mathcal{A} = (Q, Q_0, \lambda, \mathcal{X}, T, (c_i)_{1 \leq i \leq k})$ such that: Q is a finite set of locations, $Q_0 \subseteq Q$ is the set of initial locations, $\lambda \colon Q \to \mathsf{AP}$ is the labelling function, $\mathcal{X}$ is a finite set of clocks, $T \subseteq Q \times \mathcal{G}(\mathcal{X}) \times 2^{\mathcal{X}} \times Q$ is a finite set of edges, and for each $1 \leq i \leq k$, $c_i \colon Q \cup T \to \mathbb{N}$ assigns a* cost *to locations and edges.*

For $S \subseteq \mathbb{N}$, a cost c_i is said to be S-sloped *if $c_i(Q) \subseteq S$. If $S = \{0, 1\}$, it is said* stopwatch. *If $|S| = n$, we say that the cost c_i is n-sloped.*

The semantics of a weighted timed automaton $\mathcal{A}$ corresponds to the semantics of its underlying timed automaton (*i.e.*, forgetting about cost functions). It is given as a transition system $T_{\mathcal{A}} = (S, S_0, \to)$ where:

- the set of states S is $Q \times \mathbb{R}_{\geq 0}^{\mathcal{X}}$,
- the initial states are $S_0 = \{(q_0, v_0) \mid q_0 \in Q_0 \text{ and } v_0(x) = 0 \text{ for every } x \in \mathcal{X}\}$,
- the transition relation $\to$ is composed of delay and discrete moves:
  - *(delay move)* $(q, v) \xrightarrow{\tau} (q, v + \tau)$ for $(q, v) \in S$ and $\tau \in \mathbb{R}_{\geq 0}$,
  - *(discrete move)* $(q, v) \xrightarrow{e} (q', v')$ if there exists an edge $e = (q \xrightarrow{g,r} q')$ in T such that $v \models g$ and $v' = v[r \leftarrow 0]$.

  A *mixed move* $(q, v) \xrightarrow{\tau, e} (q', v')$ corresponds to the concatenation of a delay and a discrete moves $(q, v) \xrightarrow{\tau} (q, v + \tau) \xrightarrow{e} (q', v')$.

A *run* (or *execution*) in $\mathcal{A}$ is a finite path in $T_{\mathcal{A}}$, composed of mixed moves.

Let $\varrho = (q_0, v_0) \xrightarrow{\tau_1, e_1} (q_1, v_1) \xrightarrow{\tau_2, e_2} (q_2, v_2) \cdots \xrightarrow{\tau_p, e_p} (q_p, v_p)$ be a finite run in $\mathcal{A}$. For every $i \leq k$, the *i-th cost* of ϱ, denoted by $\mathsf{cost}_i(\varrho)$, is defined as:

$$\mathsf{cost}_i(\varrho) = \sum_{1 \leq j \leq p} c_i(q_{j-1}) \cdot \tau_j + \sum_{1 \leq j \leq p} c_i(e_j)$$

Informally, the cost of a run is the accumulated cost of each move along that run: delaying in some state q during d time units costs $c(q) \cdot d$, and firing an edge e costs $c(e)$. Hence, if q is a location, $c(q)$ gives the derivative of the cost function for waiting in q: we thus write $\dot{c} = 6$ on pictures. For discrete costs on transitions, we write $c{+}{=}\,3$.

Example 1. Fig. 1 models the energy consumption of a device. Due to overheating, this device cannot be left on for more than half an hour: it must either be turned to a "sleep" mode, or completely off. The model has two costs functions: cost c represents energy consumption, while cost w measures the duration of this device being on. The sleeping state consumes slightly more energy than the off state, but it is cheaper and quicker to turn the machine back on. Of course, being on consumes much more energy, but this is the only way of using the device.

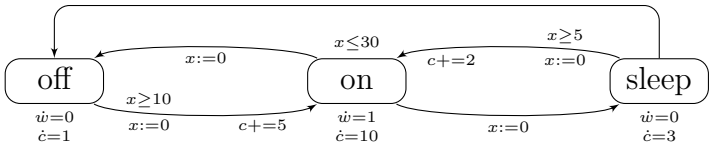


Fig. 1. Measuring the uptime of a device

2.2 The Logic WMTL

The logic WMTL is a weighted extension of LTL, but can also be viewed as an extension of MTL [Koy90], hence its name WMTL holding for "Weighted MTL".

Let $\mathcal{C}$ be a set of cost functions. We define the logic WMTL over $\mathcal{C}$ as the set of formulas defined inductively as:

$$\mathsf{WMTL} \ni \varphi \ ::= \ \sigma \ \mid \ \varphi \vee \varphi \ \mid \ \neg\varphi \ \mid \ \varphi \, \mathbf{U}_{\mathsf{cost}\sim n} \, \varphi$$

where $\sigma \in \mathsf{AP}$, cost is a cost-function symbol in $\mathcal{C}$, $\sim \in \{<, \leq, =, \geq, >\}$, and $n \in \mathbb{N}$. If there is a single cost function or if the cost function cost is clear from the context, we simply write $\varphi_1 \, \mathbf{U}_{\sim n} \, \varphi_2$ instead of $\varphi_1 \, \mathbf{U}_{\mathsf{cost}\sim n} \, \varphi_2$.

We interpret WMTL formulas over (finite) runs of weighted timed automata with $|\mathcal{C}|$ cost functions, identifying cost cost_i of $\mathcal{C}$ with the i-th cost cost_i of the runs of the automaton. Let $\mathcal{A}$ be such a weighted timed automaton, and let $\varrho = (q_0, v_0) \xrightarrow{\tau_1, e_1} (q_1, v_1) \xrightarrow{\tau_2, e_2} (q_2, v_2) \cdots \xrightarrow{e_p, \tau_p} (q_p, v_p)$ be a finite run in $\mathcal{A}$. We write $\varrho_{\geq i}$ for its suffix starting from (q_i, v_i), and $\varrho_{\leq i}$ for its prefix ending in (q_i, v_i). The satisfaction relation for WMTL is then defined inductively as follows:

$$\begin{aligned}
\varrho \models \sigma &\iff \sigma \subseteq \lambda(q_0) \\
\varrho \models \varphi_1 \vee \varphi_2 &\iff \varrho \models \varphi_1 \text{ or } \varrho \models \varphi_2 \\
\varrho \models \neg\varphi &\iff \varrho \not\models \varphi \\
\varrho \models \varphi_1 \, \mathbf{U}_{\mathsf{cost}\sim n} \, \varphi_2 &\iff \exists k > 0 \text{ s.t. } \varrho_{\geq k} \models \varphi_2, \ \forall 0 < i < k, \ \varrho_{\geq i} \models \varphi_1, \\
&\qquad\qquad \text{and } \mathsf{cost}(\varrho_{\leq k}) \sim n
\end{aligned}$$

We use classical shorthands like $\mathsf{true} \stackrel{\text{def}}{=} \sigma \vee \neg\sigma$, $\mathsf{false} \stackrel{\text{def}}{=} \neg\mathsf{true}$, $\mathbf{X} \, \varphi \stackrel{\text{def}}{=} \mathsf{false} \, \mathbf{U} \, \varphi$, $\mathbf{F}_{\mathsf{cost}\sim n} \, \varphi \stackrel{\text{def}}{=} \mathsf{true} \, \mathbf{U}_{\mathsf{cost}\sim n} \, \varphi$, and $\mathbf{G}_{\mathsf{cost}\sim n} \, \varphi \stackrel{\text{def}}{=} \neg\Big(\mathbf{F}_{\mathsf{cost}\sim n} \, \neg\varphi\Big)$.

Remark 2. Classically, there are two possible semantics for timed temporal logics [Ras99]: the continuous semantics, where the system is observed continuously, and the point-based semantics, where the system is observed only when the state of the system changes. We have chosen the latter, because the model checking problem for MTL under the continuous semantics is already undecidable [AH90].

Example 2. We continue with our previous example. Assume that the battery has been charged for a total of 1300 cost units (for cost c). We would like to know whether it is possible to use the device for at least two hours within a

period of three hours. This is equivalent to the existence of a finite path in our model satisfying the following WMTL formula:

$$\mathbf{F}_{c\leq 1300}\,\mathtt{end} \wedge \mathbf{F}_{w\geq 120}\,\mathtt{end} \wedge \mathbf{F}_{t\leq 180}\,\mathtt{end}$$

where t is a special cost measuring time, *i.e.*, $\dot{t}$ equals 1 in every location, and end characterizes the ending state of the finite path (for instance, $\mathtt{end} = \neg\mathbf{X}\,\mathsf{true}$).

2.3 The Problem and Our Results

In the following, we focus on the *existential* model-checking problem for WMTL over weighted timed automata, stated as: given $\mathcal{A}$ a weighted timed automaton and φ a WMTL formula, decide whether there exists a finite run ϱ in $\mathcal{A}$ starting in an initial state and such that $\varrho \models \varphi$. Since WMTL is closed under negation, our results obviously extend to the dual problem of *universal* model-checking.

We prove that the model-checking problem against WMTL properties is decidable for:

(1) one-clock WTAs with one stopwatch cost variable.

Any extension to that model leads to undecidability. Indeed, we prove that the model-checking problem against WMTL properties is undecidable for:

(2) one-clock WTAs with one cost variable,
(3) one-clock WTAs with two stopwatch cost variables,
(4) two-clock WTAs with one stopwatch cost variable.

We present our results as follows. In Section 3, we explain the positive result (1) using an abstraction proposed in [OW05] for proving the decidability of MTL model checking over timed automata. Then, in Section 4, we present all our undecidability results, starting with the proof for result (2), and then slightly modifying the construction for proving results (3) and (4). We conclude with some corollaries for linear hybrid automata.

3 Decidability Result

Theorem 3. *Model checking one-clock weighted timed automata with one stopwatch cost against* WMTL *properties is decidable, and non-primitive recursive.*

Proof. Time can be viewed as a special $\{1\}$-sloped cost. Hence, the non-primitive recursive lower bound follows from that of MTL model checking over finite timed words, see [OW05, OW07].

The decidability then relies on the same encoding as [OW05]. We present the construction, but don't give all details, especially when there is nothing new compared with the above-mentioned papers.

Let φ be a WMTL formula, and $\mathcal{A}$ be a single-clock weighted timed automaton with a stopwatch cost. Classically, from formula φ, we construct an "equivalent" one-variable alternating timed automaton[2] $\mathcal{B}_\varphi$. Fig. 2 displays an example of such an automaton, corresponding to formula $\mathbf{G}\,[a \Rightarrow (\mathbf{F}_{\leq 3}\, b \vee \mathbf{F}_{\geq 2}\, c)]$ (see [OW05] for more details on alternating timed automata).

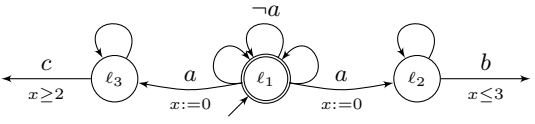


Fig. 2. A timed alternating automaton for formula $\mathbf{G}\,[a \Rightarrow (\mathbf{F}_{\leq 3}\, b \vee \mathbf{F}_{\geq 2}\, c)]$

However, note that in that case, the unique variable of the alternating automaton is not a clock but a cost variable, whose rate will depend on the location of $\mathcal{A}$ which is being visited. However, as for MTL, we have the property that $\mathcal{A} \models \varphi$ iff there is an accepting joint execution of $\mathcal{A}$ and $\mathcal{B}_\varphi$.

In the following, we write q for a generic location of $\mathcal{A}$ and ℓ for a generic location of $\mathcal{B}_\varphi$. Similarly, Q denotes the set of locations of $\mathcal{A}$ and L the set of locations of $\mathcal{B}_\varphi$.

An *$\mathcal{A}/\mathcal{B}_\varphi$-joint configuration* is a finite subset of $Q \times \mathbb{R}_{\geq 0} \cup L \times \mathbb{R}_{\geq 0}$ with exactly one element of $Q \times \mathbb{R}_{\geq 0}$ (the current state in automaton $\mathcal{A}$). The joint behaviour of $\mathcal{A}$ and $\mathcal{B}_\varphi$ is made of time evolutions and discrete steps in a natural way. Note that, from a given joint configuration γ, the time evolution is given by the current location q_γ of $\mathcal{A}$: if the cost rate in q_γ is 1, then all variables behave like clocks, *i.e.*, grow with rate 1, and if the cost rate in q_γ is 0, then all variables in $\mathcal{B}_\varphi$ are stopped, and only the clock of $\mathcal{A}$ grows with rate 1.

We encode configurations with words over the alphabet $\Gamma = 2^{(Q \times \mathsf{Reg} \cup L \times \mathsf{Reg})}$, where $\mathsf{Reg} = \{0, 1, \ldots, M\} \cup \{\top\}$ (M is an integer above the maximal constant appearing in both $\mathcal{A}$ and $\mathcal{B}_\varphi$). A state (ℓ, c) of $\mathcal{B}_\varphi$ will for instance be encoded by $(\ell, \mathit{int}(c))$ [3] if $c \leq M$, and it will be encoded by $(\ell, \top)$ if $c > M$.

Now given a joint configuration $\gamma = \{(q, x)\} \cup \{(\ell_i, c_i) \mid i \in I\}$, partition γ into a sequence of subsets $\gamma_0, \gamma_1, \ldots, \gamma_p, \gamma_\top$, such that $\gamma_\top = \{(\alpha, \beta) \in \gamma \mid \beta > M\}$, and if $i, j \neq \top$, for all $(\alpha, \beta) \in \gamma_i$ and $(\alpha', \beta') \in \gamma_j$, $\mathit{frac}(\beta) \leq \mathit{frac}(\beta')$ [4] iff $i \leq j$ (so that (α, β) and (α', β') are in the same block γ_i iff β and β' are both smaller than or equal to M and have the same fractional part). We assume in addition that the fractional part of elements in γ_0 is 0 (even if it means that $\gamma_0 = \varnothing$), and that all γ_i for $1 \leq i \leq p$ are non-empty.

If γ is a joint configuration, we define its encoding $H(\gamma)$ as the word (over Γ) $\mathsf{reg}(\gamma_0)\mathsf{reg}(\gamma_1) \ldots \mathsf{reg}(\gamma_p)\mathsf{reg}(\gamma_\top)$ where $\mathsf{reg}(\gamma_i) = \{(\alpha, \mathsf{reg}(\beta)) \mid (\alpha, \beta) \in \gamma_i\}$ with $\mathsf{reg}(\beta) = \mathit{int}(\beta)$ if $\beta \leq M$, and $\mathsf{reg}(\beta) = \top$ otherwise.

[2] We use the *eager* semantics [BMOW07] for alternating automata, where configuration of the automaton always have the same sets of successors.

[3] *int* represents the integral part.

[4] *frac* represents the fractional part.

Example 3. Consider the configuration

$$\gamma = \{(q, 1.6)\} \cup \{(\ell_1, 5.2), (\ell_2, 2.2), (\ell_2, 2.6), (\ell_3, 1.5), (\ell_3, 4.5)\}.$$

Assuming that the maximal constant (on both $\mathcal{A}$ and $\mathcal{B}_\varphi$) is 4, the encoding is then

$$H(\gamma) = \{(\ell_2, 2)\} \cdot \{(\ell_3, 1)\} \cdot \{(q, 1), (\ell_2, 2)\} \cdot \{(\ell_1, \top), (\ell_3, \top)\}$$

We define a discrete transition system over encodings of $\mathcal{A}/\mathcal{B}_\varphi$-joint configurations: there is a transition $W \Rightarrow W'$ if there exists $\gamma \in H^{-1}(W)$ and $\gamma' \in H^{-1}(W')$ such that $\gamma \to \gamma'$ (that can be either a time evolution or a discrete step).

Lemma 4. *The equivalence relation $\equiv$ defined as $\gamma_1 \equiv \gamma_2 \overset{def}{\Leftrightarrow} H(\gamma_1) = H(\gamma_2)$ is a time-abstract bisimulation over joint configurations.*

Proof. We assume that $\gamma_1 \to \gamma_1'$ and that $\gamma_1 \equiv \gamma_2$. We write $H(\gamma_1) = H(\gamma_2) = w_0 w_1 \dots w_p w_\top$ where $w_i \neq \emptyset$ if $1 \leq i \leq p$. We distinguish between the different possible cases for the transition $\gamma_1 \to \gamma_1'$.

- assume $\gamma_1 \to \gamma_1'$ is a time evolution, and the cost rate in the corresponding location of $\mathcal{A}$ is 0. If $\gamma_1 = \{(q_1, x_1)\} \cup \{(\ell_{i,1}, c_{i,1}) \mid i \in I_1\}$, then $\gamma_1' = \{(q_1, x_1 + t_1)\} \cup \{(\ell_{i,1}, c_{i,1}) \mid i \in I_1\}$ for some $t_1 \in \mathbb{R}_{\geq 0}$. We assume in addition that $\gamma_2 = \{(q_2, x_2)\} \cup \{(\ell_{i,2}, c_{i,2}) \mid i \in I_2\}$.
  We set γ_1^i the part of configuration γ_1 which corresponds to letter w_i, and we write α_1^i for the fractional part of the clock values corresponding to γ_1^i. We have $0 = \alpha_1^0 < \alpha_1^1 < \dots < \alpha_1^p < 1$. We define similarly $(\alpha_2^i)_{0 \leq i \leq p}$ for configuration γ_2. We then distinguish between several cases:
  - either $x_1 + t_1 > M$, in which case it is sufficient to choose $t_2 \in \mathbb{R}_{\geq 0}$ such that $x_2 + t_2 > M$.
  - or $x_1 + t_1 \leq M$ and $\mathit{frac}(x_1 + t_1) = \alpha_1^i$ for some $0 \leq i \leq p$. In that case, choose $t_2 = x_1 + t_1 - \alpha_1^i + \alpha_2^i - x_2$. As $\gamma_1 \equiv \gamma_2$, it is not difficult to check that $t_2 \in \mathbb{R}_{\geq 0}$. Moreover, $\mathit{frac}(x_2 + t_2) = \alpha_2^i$ and $\mathit{int}(x_2 + t_2) = \mathit{int}(x_1 + t_1)$.
  - or $x_1 + t_1 \leq M$ and $\alpha_1^i < \mathit{frac}(x_1 + t_1) < \alpha_1^{i+1}$ for some $0 \leq i \leq p$ (setting $\alpha_1^{p+1} = 1$). As previously, in that case also, we can choose $t_2 \in \mathbb{R}_{\geq 0}$ such that $\alpha_2^i < \mathit{frac}(x_2 + t_2) < \alpha_2^{i+1}$ and $\mathit{int}(x_2 + t_2) = \mathit{int}(x_1 + t_1)$.

  In all cases, defining $\gamma_2' = \{(q_2, x_2 + t_2)\} \cup \{(\ell_{i,2}, c_{i,2}) \mid i \in I_2\}$, we get that $\gamma_2 \to \gamma_2'$ and $\gamma_1' \equiv \gamma_2'$, which proves the inductive case.
- there are two other cases (time evolution with rate of all variables being 1, and discrete step), but they are similar to the case of MTL, and we better refer to [OW07]. □

Hence, from the previous lemma, we get:

Corollary 5. *$W \Rightarrow^* W'$ iff there exist $\gamma \in H^{-1}(W)$ and $\gamma' \in H^{-1}(W')$ such that $\gamma \to^* \gamma'$.*

The set $\Gamma = 2^{(Q\times\mathsf{Reg}\cup L\times\mathsf{Reg})}$ is naturally ordered by inclusion $\subseteq$. We extend the classical subword relation for words over Γ as follows: Given two words $a_0a_1\dots a_n$ and $a'_0a'_1\dots a'_{n'}$ in Γ^*, we say that $a_0a_1\dots a_n \sqsubseteq a'_0a'_1\dots a'_{n'}$ whenever there exists an increasing injection $\iota : \{0,1,\dots,n\} \to \{0,1,\dots,n'\}$ such that for every $i \in \{0,1,\dots,n\}$, $a_i \subseteq a'_{\iota(i)}$. Following [AN00, Theorem 3.1], the preorder $\sqsubseteq$ is a well-quasi-order.

Lemma 6. *Assume that $W_1 \sqsubseteq W_2$, and that $W_2 \Rightarrow^* W'_2$. Then, there exists $W'_1 \sqsubseteq W'_2$ such that $W_1 \Rightarrow^* W'_1$.*

The algorithm then proceeds as follows: we start from the encoding of the initial configuration, say W_0, and then generate the tree unfolding of the implicit graph $(\Gamma^*, \Rightarrow)$, stopping a branch when the current node is labelled by W such that there already exists a node of the tree labelled by W' with $W' \sqsubseteq W$ (note that by Lemma 6, if there is an accepting path from W, then so is there from W', hence it is correct to prune the tree after node W). Note that this tree is finitely branching. Hence, if the computation does not terminate, then it means that there is an infinite branch (by König lemma). This is not possible as $\sqsubseteq$ is a well-quasi-order. Hence, the computation eventually terminates, and we can decide whether there is a joint accepting computation in $\mathcal{A}$ and $\mathcal{B}_\varphi$, which implies that we can decide whether $\mathcal{A}$ satisfies φ or not. □

Remark 7. In the case of MTL, the previous encoding can be used to prove the decidability of model checking for timed automata with any number of clocks. In our case, it cannot: Lemma 4 does not hold for two-clock weighted timed automata, even with a single stopwatch cost. Consider for instance two clocks x and z, and a cost variable cost. Assume we are in location q of the automaton with cost rate 0 and that there is an outgoing transition labelled by the constraint $x = 1$. Assume moreover that the value of z is 0, whereas the value of x is 0.2. We consider two cases: either the value of cost is 0.5, or the value of cost is 0.9. In both cases, the encoding[5] of the joint configuration is $\{(q,z,0)\} \cdot \{(q,x,0)\} \cdot \{(\mathsf{cost},0)\}$. However, in the first case, the encoding when firing the transition will be $\{(q,x,1)\} \cdot \{(\mathsf{cost},0)\} \cdot \{(q,z,0)\}$, whereas in the second case, it will be $\{(q,x,1)\} \cdot \{(q,z,0)\} \cdot \{(\mathsf{cost},0)\}$. Hence the relation $\equiv$ is not a time-abstract bisimulation.

4 Undecidability Results

4.1 One-Clock WTAs with One Cost Variable

Theorem 8. *Model checking one-clock weighted timed automata with one cost variable against* WMTL *properties is undecidable.*

We push some ideas used in [BBM06, BLM07] further to prove this new undecidability result. We reduce the halting problem for a two-counter machine $\mathcal{M}$

[5] We extend the encoding we have presented above to several clocks, as originally done in [OW05].

to that problem. The unique clock of the automaton will store both values of the counters. If the first (resp. second) counter has value c_1 (resp. c_2), then the value of the clock will be $2^{-c_1}3^{-c_2}$. Our machine $\mathcal{M}$ has two kinds of instructions. The first kind increments one of the counter, say $\mathtt{c}$, and jumps to the next instruction:

$$\mathtt{p_i}: \ \mathtt{c} := \mathtt{c} + 1; \ \mathtt{goto}\ \mathtt{p_j}. \tag{1}$$

The second kind decrements one of the counter, say $\mathtt{c}$, and goes to the next instruction, except if the value of the counter was zero:

$$\mathtt{p_i}: \ \mathtt{if}\ (\mathtt{c} == 0)\ \mathtt{then\ goto}\ \mathtt{p_j}\ \mathtt{else}\ \mathtt{c} := \mathtt{c} - 1; \ \mathtt{goto}\ \mathtt{p_k}. \tag{2}$$

Our reduction consists in building a weighted timed automaton $\mathcal{A}_\mathcal{M}$ and a WMTL formula φ such that the two-counter machine $\mathcal{M}$ halts iff $\mathcal{A}_\mathcal{M}$ has an execution satisfying φ. Each instruction of $\mathcal{M}$ is encoded as a module, all the modules are then plugged together.

Module for Instruction (1). Consider instruction (1), which increments the first counter. To simulate this instruction, we need to be able to divide the value of the clock by 2. The corresponding module, named Mod_i, is depicted on Fig. 3.[6]

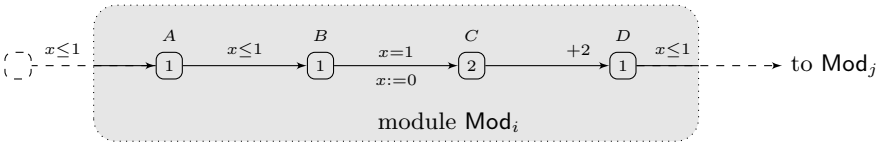


Fig. 3. Module for incrementing c_1

The following lemma is then easy to prove:

Lemma 9. *Assume that there is an execution ϱ entering module Mod_i with $x = x_0 \leq 1$, exiting with $x = x_1$, and such that no time elapses in A and D and the cost between A and D equals 3. Then $x_1 = x_0/2$.*

A similar result can be obtained for a module incrementing c_2: it simply suffices to replace the cost rate in C by 3 instead of 2.

Module for Instruction (2). Consider instruction (2). The simulation of this instruction is much more involved than the previous instruction. Indeed, we first have to check whether the value of x when entering the module is of the form 3^{-c_2} (*i.e.*, whether $c_1 = 0$). This is achieved, roughly, by multiplying the value of x by 3 until it reaches (or exceeds) 1. Depending on the result, this module will then branch to module Mod_j or decrement counter c_1 and go to module Mod_k. The difficult point is that clock x must be re-set to its original value between the first and the second part. We consider the module Mod_i depicted on Fig. 4.

[6] As there is a unique cost variable, we write its rate within the location, and add a discrete incrementation (*eg* +2) on edges, when the edge has a positive cost.

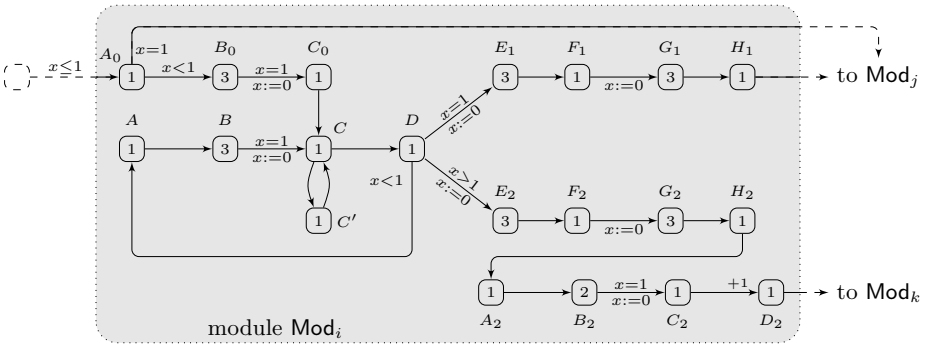


Fig. 4. Module testing/decrementing c_1

Lemma 10. *Assume there exists an execution ϱ entering module* Mod$_i$ *with $x = x_0 \leq 1$, exiting to module* Mod$_j$ *with $x = x_1$, and such that*

- *no time elapses in A_0, C_0, D, A, C', F_1 and H_1;*
- *any visit to C_0 or C' is eventually followed (strictly) by a visit to C' or F_1;*
- *the cost exactly equals 3 along each part of ϱ between A or A_0 and the next visit in D, between C_0 or C' and the next visit in C' or F_1, and between the last visit to D and H_1.*

Then $x_1 = x_0$ and there exists $n \in \mathbb{N}$ s.t. $x_0 = 3^{-n}$.

Proof. Let ϱ be such an execution. First, if $x_0 = 1$ and ϱ goes directly to module Mod$_j$, then the result immediately follows.

Otherwise, ϱ visits D at least once. We prove inductively that, at the k-th visit in D, the value of x equals $3^k x_0$ (remember that no time can elapse in D). The first part of ϱ between A_0 and D is as follows[7] (the labels on the arrows represent the cost of the corresponding transition):

$$(A_0, x_0) \xrightarrow{0} (B_0, x_0) \xrightarrow{3(1-x_0)} (B_0, 1) \xrightarrow{0} (C_0, 0) \xrightarrow{0} (C, 0) \xrightarrow{\alpha} (C, \alpha) \xrightarrow{0} (D, \alpha).$$

The total cost, $3(1 - x_0) + \alpha$, must equal 3. Thus $\alpha = 3x_0$. A similar argument shows that one turn in the loop (from D back to itself) also multiplies clock x by 3, hence the result. Since ϱ eventually fires the transition from D to E_1, it must be the case that $x_0 = 3^{-n}$ for some $n \in \mathbb{N}$.

We now prove that $x_1 = x_0$. The proof follows a similar line: we prove that at the k-th visit to C_0 or C', the value of x is $(3^k - 3)x_0$. This clearly holds when $k = 1$ (*i.e.*, when we visit C_0). Assuming that ϱ eventually visits C', we consider the part of ϱ between C_0 and the first visit to C':

$$(C_0, 0) \xrightarrow{0} (C, 0) \xrightarrow{3x_0} (C, 3x_0) \xrightarrow{0} (D, 3x_0) \xrightarrow{0} (A, 3x_0) \xrightarrow{0} (B, 3x_0)$$
$$(B, 3x_0) \xrightarrow{3(1-3x_0)} (B, 1) \xrightarrow{0} (C, 0) \xrightarrow{\beta} (C, \beta) \xrightarrow{0} (C', \beta).$$

[7] By contradiction, it can be proved that C' cannot be visited along that part of ϱ, since the cost between C_0 and C' must be exactly 3.

The cost of this part is $3-6x_0+\beta$, and must equal 3. Thus $\beta = 6x_0$ as required. A similar computation (considering each part of ϱ between two consecutive visits to C') proves the inductive case.

Now, consider the part from the last visit of C' to H_1:

$$(C', (3^n-3)x_0) \xrightarrow{0} (C, (3^n-3)x_0) \xrightarrow{3x_0} (C, 3^n x_0) \xrightarrow{0} (D, 3^n x_0) \xrightarrow{0} (E_1, 0)$$
$$(E_1, 0) \xrightarrow{3\gamma} (E_1, \gamma) \xrightarrow{0} (F_1, \gamma) \xrightarrow{0} (G_1, 0) \xrightarrow{3\delta} (G_1, \delta) \xrightarrow{0} (H_1, \delta).$$

Remember that $3^n x_0 = 1$, which explains why the computation goes to E_1 instead of E_2). The cost between C' and F_1 is $3x_0 + 3\gamma$, and equals 3. Thus $\gamma = 1 - x_0$. Similarly, the cost between D and H_1 is $3\gamma + 3\delta$ and must equal 3, which proves that δ, which is precisely x_1, equals x_0. □

We have a similar result for a trajectory going to module Mod_k:

Lemma 11. *Assume there exists an execution ϱ entering module Mod_i with $x = x_0 \leq 1$, exiting to module Mod_k with $x = x_1$, and such that*

- *no time elapses in A_0, C_0, D, A, C', F_2 H_2, A_2 and D_2;*
- *any visit to C_0 or C' is eventually followed (strictly) by a visit to C' or F_2;*
- *the cost exactly equals 3 along each part of ϱ between A or A_0 and the next visit in D, between C_0 or C' and the next visit in C' or F_2, between D and H_2, and between H_2 and D_2.*

Then $x_1 = 2x_0$ and for every $n \in \mathbb{N}$, $x_0 \neq 3^{-n}$.

Proof. The arguments of the previous proof still apply: the value of x at the k-th visit to D is $3^k x_0$. If x_0 had been of the form 3^{-n}, then ϱ would not have been able to fire the transition to E_2. Also, the value of x when ϱ visits H_2 is precisely x_0. The part from H_2 to D is then as follows:

$$(H_2, x_0) \xrightarrow{0} (A_2, x_0) \xrightarrow{0} (B_2, x_0) \xrightarrow{2(1-x_0)} (B_2, 1) \xrightarrow{0} (C_2, 0) \xrightarrow{\kappa} (C_2, \kappa) \xrightarrow{1} (D_2, \kappa).$$

The cost of this part is $2(1-x_0)+\kappa+1$, so that $x_1 = \kappa = 2x_0$. □

Again, these results can easily be adapted to the case of an instruction testing and decrementing c_2: it suffices to

- set the costs of states B_0, B, E_1, E_2, G_1 and G_2 to 2,
- set the cost of B_2 to 3,
- set the discrete cost of $C_2 \to D_2$ to 0
- set the discrete costs of $C \to D$, $G_1 \to H_1$ and $G_2 \to H_2$ to +1.

Global Reduction. We now explain the global reduction: the automaton $\mathcal{A}_\mathcal{M}$ is obtained by plugging the modules above following the instructions of $\mathcal{M}$. There is one special module for instruction Halt, which is made of a single Halt state. We also add a special initial state that lets 1 t.u. elapse (so that $x = 1$) before entering the first module.

The WMTL formula is built as follows: we first define an intermediary subformula stating that no time can elapse in some given state. It writes $\mathsf{zero}(P) = \mathbf{G}\,(P \Rightarrow (P\,\mathbf{U}_{=0}\,\neg P))$. If the local cost in state P is not zero (which is the case in all the states of $\mathcal{A}_{\mathcal{M}}$), this formula forbids time elapsing in P. We then let φ_1 be the formula requiring that time cannot elapse in a state labelled with A, D, A_0, C_0, C', F_1, F_2 H_1, H_2, A_2 and D_2. It remains to express the second and third conditions in Lemmas 9, 10 and 11. This is the role of the following formula φ_2:

$$\begin{aligned}\varphi_2 = \mathbf{G}\,[(A \vee A_0) \Rightarrow (\neg D\,\mathbf{U}_{=3}\,D)]\wedge\\ \mathbf{G}\,[(C_0 \vee C') \Rightarrow (\neg(C' \vee F_1)\,\mathbf{U}_{=3}\,(C' \vee F_1))]\wedge\\ \mathbf{G}\,[D \Rightarrow ((\neg A\,\mathbf{U}\,A) \vee (\neg(H_1 \vee H_2)\,\mathbf{U}_{=3}\,(H_1 \vee H_2)))]\wedge\\ \mathbf{G}\,[H_2 \Rightarrow (\neg D_2\,\mathbf{U}_{=3}\,D_2)].\end{aligned}$$

The following proposition is now straightforward:

Proposition 12. *The machine $\mathcal{M}$ halts iff there exists an execution in $\mathcal{A}_{\mathcal{M}}$ satisfying $\varphi_1 \wedge \varphi_2 \wedge \mathbf{F}\,\mathsf{Halt}$.*

Remark 13.
- For the sake of simplicity, our reduction uses discrete costs, so that our WMTL formulas only involve constraints "$= 0$" and "$= 3$" (and the same formula φ_2 can be used for both counters). But our undecidability result easily extends to automata without discrete costs.
- Our reduction uses a $\{1, 2, 3\}$-sloped cost variable, but it could be achieved with any $\{p, q, r\}$-sloped cost variable (with $0 < p < q < r$, and p, q and r are pairwise coprime) by encoding the values of the counters by the clock value $(p/q)^{c_1} \cdot (p/r)^{c_2}$.
- Our WMTL formula can easily be turned into a WMITL formula (whose syntax is that of MITL [AFH96], *i.e.*, with no punctual constraints). It suffices to replace formulas of the form $(\neg p)\,\mathbf{U}_{=n}\,p$ with $(\neg p)\,\mathbf{U}_{\leq n}\,p \wedge (\neg p)\,\mathbf{U}_{\geq n}\,p$.

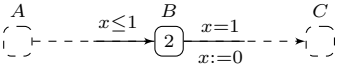
4.2 Two-Clock WTAs with One Stopwatch-Cost Variable

We now prove a similar result for WTAs with two clocks but only with a stopwatch cost (*i.e.*, a cost with slope 0 or 1).

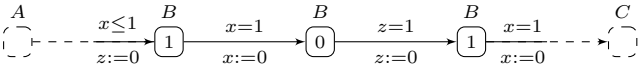
Theorem 14. *Model checking two-clock weighted timed automata with one stopwatch cost against* WMTL *properties is undecidable.*

Proof (Sketch). The proof uses the same encoding, except that states with cost 2 or 3 are replaced by sequences of states with costs 0 and 1 having the same effect. We have two different kinds of states with cost 2 (or 3):

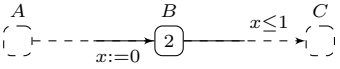
- those in which we stay until $x = 1$:



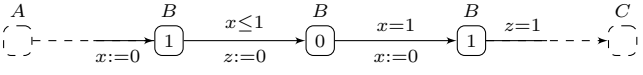
These states are replaced by the following submodule:



– those in which we enter with $x = 0$ (and exit with $x \leq 1$):



Those are replace with a slightly different sequence of states:



Those transformations are easily adapted to states with cost 3. □

4.3 One-Clock WTAs with Two Stopwatch-Cost Variables

In the above constructions, each clock can be replaced with an observer variable, *i.e.*, with a "clock cost" that is not involved in the guards of the automaton anymore. We briefly explain this transformation on an example, and leave the details to the keen reader.

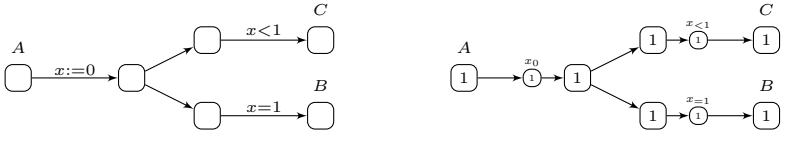


Fig. 5. Replacing a clock with an extra "clock cost"

Fig. 5 displays the transformation to be applied to the automaton. It then suffices to enforce that no time elapses in states x_0, $x_{<1}$ and $x_{=1}$, and that the following formula holds:

$$\bigwedge_{\sim n \in \{<1,=1\}} \mathbf{G}\left[\left(x_0 \wedge \neg x_0 \,\mathbf{U}\, x_{\sim n}\right) \Rightarrow \left(\neg x_0 \,\mathbf{U}_{(c_x \sim n)}\, x_{\sim n}\right)\right]$$

This precisely encodes the role of clock x in the original automaton with a clock cost, which is in particular a stopwatch cost. Note that this transformation is not correct in general, but it is here because our reduction never involves two consecutive transitions with the same guard. As a consequence:

Theorem 15. – *Model checking one-clock weighted timed automata with two stopwatch costs against* WMTL *properties is undecidable.*
– *Model checking zero-clock weighted timed automata with two costs (or three stopwatch costs) against* WMTL *properties is undecidable.*

4.4 Remarks on Hybrid Systems

It is worth making some remarks on the relation between our undecidability results and the undecidability of reachability in linear hybrid automata (LHA for short; we refer to [HKPV98] for their definition). Indeed, the construction on Fig. 5 is not surprising as MITL can capture the exact behaviour of timed automata [Ras99], and could thus be used to transfer undecidability results from LHA to WMTL model checking: from an LHA $\mathcal{H}$, we can construct a WMTL formula $\varphi_{\mathcal{H}}$ such that a given location of $\mathcal{H}$ is reachable iff the WTA resulting from the transformation satisfies formula $\varphi_{\mathcal{H}}$.

To the best of our knowledge, the tightest undecidability results for LHA hold either with five clocks and a single two-sloped variable [HKPV98], or with four stopwatch variables [Fle02].

From these results and the remarks above, the model checking of WMTL is undecidable for WTAs, either with five clocks and a single two-sloped cost variable, or without clocks and four stopwatch costs. While the first result cannot be compared with any of our results, the second is subsumed by Theorem 15.

On the other hand, by using similar ideas to those we have developed in this paper, we can get noticeable undecidability results for LHA: it suffices to encode as an LHA all the constraints mentioned in Lemmas 9, 10, 11, which can be done rather easily using two variables and one clock. However, we can do better and adapt the construction to get the undecidability already with one variable and two clocks. We present this construction now, which can be interesting in itself to understand the power of LHA. The incrementation instruction follows rather easily from that of Fig. 3, we better present the modified construction

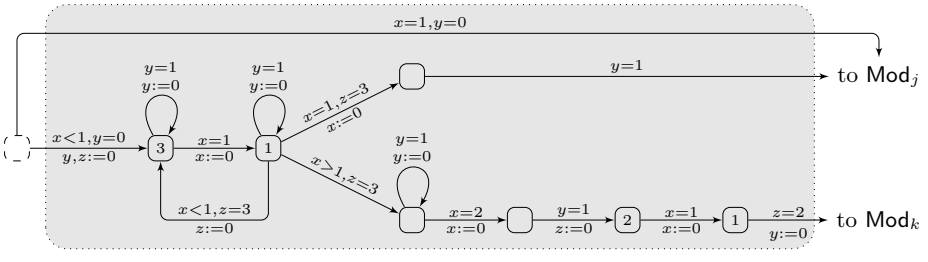


Fig. 6. Undecidability of LHA with two clocks (x, y) and a single hybrid variable (z)

for the test and decrementation instruction (see Fig. 6, where the derivative of variable z, when relevant, is indicated in each location). It differs from Fig. 4 in the way the initial value of clock x is recovered at the end of the loop which checks whether the value of clock x is of the form 3^{-n} or not: clock y is reset when entering the module and then computes a modulo 1, which implies that the initial value of clock x, say x_0, is recovered each time $y = 1$.

Corollary 16. *The reachability problem for LHA, either with two clocks and a single $\{p, q, r\}$-sloped variable (with p, q and r positive and pairwise coprime), or with three clocks and a single stopwatch variable, is undecidable.*

Finally, note that the class of LHA with one clock and one stopwatch variable can easily be proved decidable using the classical regions of [AD94].

References

[AD94] Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)

[AFH96] Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM 43(1), 116–146 (1996)

[AH90] Alurand, R., Henzinger, T.A.: Real-time logics: Complexity and expressiveness. In: Proc. 5th Annual Symposium on Logic in Computer Science (LICS'90), pp. 390–401. IEEE Computer Society Press, Los Alamitos (1990)

[ALP01] Alur, R., Torre, S.L., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)

[AN00] Abdulla, P.A., Nylén, A.: Better is better than well: On efficient verification of infinite-state systems. In: Proc. 15th Annual Symposium on Logic in Computer Science (LICS'00), pp. 132–140. IEEE Computer Society press, Los Alamitos (2000)

[BBBR07] Bouyer, P., Brihaye, T., Bruyère, V., Raskin, J.-F.: On the optimal reachability problem. Formal Methods in System Design (to appear, 2007)

[BBL04] Bouyer, P., Brinksma, E., Larsen, K.G.: Staying alive as cheaply as possible. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 203–218. Springer, Heidelberg (2004)

[BBM06] Bouyer, P., Brihaye, T., Markey, N.: Improved undecidability results on weighted timed automata. Information Processing Letters 98(5), 188–194 (2006)

[BBR04] Brihaye, T., Bruyère, V., Raskin, J.-F.: Model-checking for weighted timed automata. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 277–292. Springer, Heidelberg (2004)

[BBR05] Brihaye, T., Bruyère, V., Raskin, J.-F.: On optimal timed strategies. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 49–64. Springer, Heidelberg (2005)

[BBR06] Brihaye, T., Bruyère, V., Raskin, J.-F.: On model-checking timed automata with stopwatch observers. Information and Computation 204(3), 447–478 (2006)

[BFH+01] Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.: Minimum-cost reachability for priced timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)

[BLM07] Bouyer, P., Larsen, K.G., Markey, N.: Model-checking one-clock priced timed automata. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 108–122. Springer, Heidelberg (2007)

[BLMR06] Bouyer, P., Larsen, K.G., Markey, N., Rasmussen, J.I.: Almost optimal strategies in one-clock priced timed automata. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 345–356. Springer, Heidelberg (2006)

[BMOW07] Bouyer, P., Markey, N., Ouaknine, J., Worrell, J.: The cost of punctuality. In: Proc. 21st Annual Symposium on Logic in Computer Science (LICS'07). IEEE Computer Society Press (to appear, 2007)

[Fle02] Fleury, E.: Automates temporisés avec mises à jour. PhD thesis, École Normale Supérieure de Cachan, Cachan, France (2002)

[HKPV98] Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? Journal of Computer and System Sciences 57(1), 94–124 (1998)

[Koy90] Koymans, R.: Specifying real-time properties with Metric Temporal Logic. Real-Time Systems 2(4), 255–299 (1990)

[LMS04] Laroussinie, F., Markey, N., Schnoebelen, P.: Model checking timed automata with one or two clocks. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 387–401. Springer, Heidelberg (2004)

[LR05] Larsen, K.G., Rasmussen, J.I.: Optimal conditional scheduling for multipriced timed automata. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 234–249. Springer, Heidelberg (2005)

[LW05] Lasota, S., Walukiewicz, I.: Alternating timed automata. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 250–265. Springer, Heidelberg (2005)

[LW07] Lasota, S., Walukiewicz, I.: Alternating timed automata. ACM Transactions on Computational Logic (to appear, 2007)

[OW05] Ouaknine, J., Worrell, J.: On the decidability of Metric Temporal Logic. In: Proc. 19th Annual Symposium on Logic in Computer Science (LICS'05), pp. 188–197. IEEE Computer Society Press, Los Alamitos (2005)

[OW06] Ouaknine, J., Worrell, J.: On Metric Temporal Logic and faulty Turing machines. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006 and ETAPS 2006. LNCS, vol. 3921, pp. 217–230. Springer, Heidelberg (2006)

[OW07] Ouaknine, J., Worrell, J.: On the decidability and complexity of Metric Temporal Logic over finite words. Logical Methods in Computer Science 3(1:8), 1–27 (2007)

[Ras99] Raskin, J.-F.: Logics, Automata and Classical Theories for Deciding Real-Time. PhD thesis, University of Namur, Namur, Belgium (1999)

Hypervolume Approximation in Timed Automata Model Checking

Víctor Braberman[1,2], Jorge Lucángeli Obes[1], Alfredo Olivero[3], and Fernando Schapachnik[1]

[1] Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina*
{vbraber,lucangeli,fschapac}@dc.uba.ar
[2] CONICET
[3] Facultad de Ingeniería y Cs. Exactas,
Universidad Argentina de la Empresa, Buenos Aires, Argentina**
aolivero@uade.edu.ar

Abstract. Difference Bound Matrices (DBMs) are the most commonly used data structure for model checking timed automata. Since long they are being used in successful tools like KRONOS or UPPAAL.

As DBMs represent convex polyhedra in an n-dimensional space, this paper explores the idea of using its hypervolume as the basis for two optimization techniques. One of them is very simple to implement. The other, an improvement over the first, requires more involved programming. Each of them saves verification time (up to 19% in our case studies), with a modest increase of memory requirements. Their impact differs among the different case studies but, as they can be combined, there is no need to choose a priori.

1 Introduction

In current days timed systems are both pervasive and critical, ranging from embedded and PDAs to plant and flight controllers. Their complexity is ever increasing so unassisted ways of verifying them make sense. Automated methods, however, are known to suffer from scalability problems: their time and memory requirements grow exponentially as systems increase in size. This is why any technique that can palliate such problems can be useful.

We focus on model checking of timed automata [1], an extension of finite automata with the possibility to model dense time. Many model checking property validation problems can be reduced to forward reachability [2], that is, an exploration of the model starting by its initial state and trying to find a state tagged with a particular boolean property, call it p. The basic (conceptual) procedure is straight forward: insert the initial state in the *Pending* queue and initialize an empty *Visited* set. Then, while *Pending* is not empty, take its next state

* Partially supported by UBACyT 2004 X020, PICT 32440 and 11738.
** Partially supported by STIC-AmSud project Tapioca.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 69–81, 2007.

and check whether the property p holds for it. If it does, finish with a "YES" result, otherwise, put it in *Visited*, compute its (timed) successors (one for each outgoing transition). To ensure termination, before putting them in *Pending*, it should be checked that they are not *included* in any other state from *Visited*. In an untimed exploration the check would be simpler: is the same state already *present* in *Visited*? In the timed (symbolic) framework, clocks values conform multi-dimensional polyhedra. As such, if a new state is included in an already explored one there is no point in revisiting it, even if it is not exactly equal. Some more detail on the reachability procedure is presented in Fig. 1, on page 73.

From the outline of the reachability algorithm it should be clear that the inclusion operation between polyhedra is a critical one, being responsible for an important fraction of the total running time. As such, finding ways to speed it up is always a good idea.

Although many optimizations have been developed (see, for example, [3,4,5,6,7,8,9]), the inclusion checking operation, although linear in many cases, has a worst case of $O(n^2)$ where n is the number of clocks in the system. In this article we focus on the hypervolume of the above mentioned polyhedra. If it could be easily computed, an $O(1)$ check could be performed prior to the expensive inclusion algorithm: if $hypervolume(A) > hypervolume(B)$ it is impossible for A to be included in B. Of course, if the hypervolume of A is less or equal to B's, then A can be included or not, and the full check needs to be performed.

Computing exact hypervolume for n-dimensional polyhedra is a known hard problem, but luckily with the traditional data structures used for timed model checking (Difference Bound Matrices) an approximation can be computed very cheaply. This approximation is safe, in the sense that the observations from the previous paragraph still hold (Theorem 1 expresses the property formally). A related idea might be comparing the bounding box in each dimension. It has the advantage of being less suceptible to overflow (see Section 3.2), but that would require $O(n)$ extra storage and comparisons, instead of $O(1)$.

We take advantage of the hypervolume approximation in two ways: firstly, inclusion checks are avoided in the sense of the preceding paragraphs. Secondly, the *Visited* set is ordered by (approximate) hypervolume. In this way, it is not necessary to check a new state against the complete set, but only against the states that have a bigger (approximate) hypervolume, saving an important number of inclusion checks (see Section 4).

Neither technique increments the number of visited states. They can be used independently and obtain interesting speedups. Some models benefit more with one of them, and some with the other, with acceleration of up to 19% in our experiments. They can also be combined, although the speedups are not additive, because there is some mutual cancelation. The good news is that there is no need to speculate on which one to use, because using both is generally as good as using the best of them.

Further, we explain why it doesn't makes sense to order also *Pending* by its hypervolume.

Although the idea of approximations is not new (see, for instance the convex-hull abstraction at [10]), they generally lead to approximated answers also (i.e., they might state with certainty that the property is not reached, or that it might –or not– be reached). Ours, however, gives exact answers.

The rest of the article is structured as follows: Section 2 gives the basic background on the timed automata formalism and related definitions. The following section presents the details of the proposed techniques. In Section 4 empirical evidence, based on known case studies from the literature, is presented. After that, Section 5 discusses future work and concludes the article.

2 Background

Timed automata [1] are a widely used formalism to model and analyze timed systems. They are supported by several tools such as KRONOS [11] or UPPAAL [12]. Their semantics are based on labeled state-transition systems and time-divergent runs over them. Here we present their basic notions and refer the reader to [1,11] for a complete formal presentation.

Definition 1 (Timed automaton). *A* timed automaton *(TA) is a tuple* $\mathcal{A} = \langle L, X, \Sigma, E, I, \mathsf{l}_0 \rangle$*, where* L *is a finite set of locations,* X *is a finite set of clocks (non-negative real variables),* Σ *is a set of labels,* E *is a finite set of edges,* $I : L \overset{tot}{\rightarrow} \Psi_X$ *is a total function associating to each location a clock constraint called the location's invariant, and* $\mathsf{l}_0 \in L$ *is the initial location. Each edge in* E *is a tuple* $\langle \mathsf{l}, a, \psi, \alpha, \mathsf{l}' \rangle$*, where* $\mathsf{l} \in L$ *is the source location,* $\mathsf{l}' \in L$ *is the target location,* $a \in \Sigma$ *is the label,* $\psi \in \Psi_X$ *is the guard,* $\alpha \subseteq X$ *is the set of clocks reset at the edge. The set of clock constraints* Ψ_X *for a set of clocks* X *is defined according to the following grammar:* $\Psi_X \ni \psi ::= x \sim c | \psi \wedge \psi$*, where* $x \in X, \sim \in \{<, \leq, =, >, \geq\}$ *(although invariants restrict* $\sim$ *to* $\{<, \leq\}$*) and* $c \in \mathbb{N}$*.*

Usually, a TA $\mathcal{A}$ has an associated mapping $Pr : L \mapsto 2^{Props}$ which assigns to each location a subset of propositional variables from the set $Props$.

The parallel composition $\mathcal{A}_1 \parallel \mathcal{A}_2$ of TAs $\mathcal{A}_1$ and $\mathcal{A}_2$ is defined using a label-synchronized product of automata [1,11]. At any time, the *state* of the system is determined by the location and the values of clocks, which must satisfy the location invariant. The system can evolve in two different ways: either an enabled transition is taken, changing the location and (maybe) resetting some clocks while the others keep their values unaltered (a discrete step), or it may let some amount of time pass (a timed step). In the last case the system remains in the same location and all clocks increase according to the elapsed time, while still satisfying the location invariant.

To model a complex system, an automaton can be expressed as the parallel composition of the automata representing each component. A location of the obtained automaton, called *global location* or *composed automata node*, is a tuple consisting of a location of each component. Similarly, a state of the automaton (*global state*) is a global location plus the values of all clocks. Such sets of automata are usually called *a network*.

Definition 2 (Clock valuations). *A* valuation *is a total function from the clock set X into $\mathbb{R}^+$ (i.e., the reading of each clock in a particular moment). The valuation set over X, $\mathcal{V}_X$ is defined as $\{v : X \overset{tot}{\rightarrow} \mathbb{R}^+\}$. For each $v \in \mathcal{V}_X$ and $\delta \in \mathbb{R}^+$, $v+\delta$ stands for the valuation defined as $(\forall x \in X)(v+\delta)(x) = v(x)+\delta$.*

To deal with infinite state manipulation, convex sets of clock valuations are symbolically represented as conjunctions of inequalities (e.g., $1 \leq x \leq 5 \wedge x - y > 8$). Each of these conjunction represents a convex set of points, and is referred to as a *zone*. A data structure called Difference Bound Matrices (DBM) [13] is typically used to manipulate such kind of information.

Definition 3 (Difference Bound Matrices). *DBMs are $(n+1)^2$ matrices, where n is the number of clocks in the system. Diagonal cells are void, but all the others contain a tuple $\langle \prec, c \rangle$ called* bound*, where $\prec \in \{<, \leq\}$ and $c \in \mathbb{Z} \cup \{\infty\}$. If in a DBM M cell (i, j) (noted $M[i, j]$) contains $\langle \prec, c \rangle$ it means that $x_i - x_j \prec c$, where x_i and x_j are the i-th and j-th clocks in the system (counting 0 as a special clock, used to express $x_i \prec c$ as $x_i - 0 \prec c$). The only valid use of ∞ in a cell is to mean that the difference between two clocks is unbounded. I.e., $x_i - x_j < \infty$.*

Although a zone *can be represented by a* DBM*, we will abuse both terms using them interchangeably when there is no confusion.*

During the reachability algorithm (see Fig. 1), states are represented by a pair (l, z) where l is a location and z a timed zone. Given a state, the successor set is computed by the $suc_{\triangleright}$ operator, which is defined as follows:

$$suc_{\triangleright}(\mathsf{l}, z) = \{(\mathsf{l}', z') / \langle \mathsf{l}, a, \psi, \alpha, \mathsf{l}' \rangle \in E \wedge z' = suc_{\tau}(reset_{\alpha}(z \cap \psi)) \cap I(\mathsf{l}')\}$$

Where $reset_{\alpha}$ means putting the clocks in α to zero and $suc_{\tau}(\psi)$ means replacing the constraints of the form $x \prec c$ by $x < \infty$ while leaving the rest intact. To avoid clutter, calls to *cf*, the *canonical form* function, are not shown. It expresses every constraint as tight as possible (see, for instance, [14], for the details) and should be called after intersection, reset and suc_{τ}.

Not every constraint needs to be present for all the operations. Actually, a reduced version of the constraint systems can be used for most operations, thus saving memory [15]. In practice, most tools use a variation of DBMs, called *Minimal Constraint Representation*, which employs that idea. As these DBMs are sparse, they are not stored like proper matrices, but as a linked list of constraints, in order to save space. This minimal representation sets the context of this article.

The classical algorithm for inclusion checking in such data structure is shown in Fig. 2. It expects the first zone to be in canonical form and the second one to be *minimized*, i.e., as the above mentioned Minimal Constraint Representation. If a different representation was chosen for the DBMs, the algorithms would change importantly.

This algorithm is easily generalized to check if a zone is included in a set of zones (IsIncludedInSet).

```
1: function ForwardReach(Property φ)
2:     Visited ← ∅
3:     Pending ← {(l_0, z_0)}
4:     while Pending ≠ ∅ do
5:         (l, z) ← next(Pending)
6:         Add((l, z), Visited)
7:         if (l, z) ⊨ φ then return YES
8:         end if
9:         Z_l ← ⋃_{(l,z')∈(Visited∪Pending)} z'
10:        for (l', z') ∈ suc_▷(l, z) do
11:            if ¬ IsIncludedInSet(z', Z_l) then
12:                Add((l', z'), Pending)
13:            end if
14:            if ∃z'' ∈ Pending_l/ IsIncluded(z'', z') then
15:                Delete((l', z''), Pending)
16:            end if
17:        end for
18:    end while
19:    return NO
20: end function
```

Fig. 1. Forward reachability algorithm

```
1: function IsIncluded(DBM z_1, z_2)
2:     // Require: z_1 is in canonical form, z_2 is minimized.
3:     for i = 0 to #clocks do
4:         for j = 0 to #clocks do
5:             if i ≠ j ∧ ¬(z_1[i, j] ≤ z_2[i, j]) then
6:                 return NO
7:             end if
8:         end for
9:     end for
10:    return YES
11: end function
```

Fig. 2. Inclusion checking algorithm

3 Using the Hypervolume

3.1 Avoiding Checks by Comparing Hypervolume Approximations

Let's start by defining which approximation to the hypervolume we use. The idea is to compute the hypervolume of the smallest hypercube containing the polyhedron defined by the clocks' values.

Definition 4 (Hypervolume approximation). *z be a DBM with n clocks. $HVol(z)$ is defined as $\Pi_{1\leq i\leq n}(const(z[i,0]) - |const(z[0,i])|)$, where $const(x \prec c) = c$.*

Note that for the purpose of the hypervolume approximation there is no need to differentiate among $\prec$, and that const$(z[0,i])$, the lower bound, is negative and thus the need for modulus (i.e., $x_1 > 7$ is expressed in DBMs as $z[0,1] = (<, -7)$).

If $z[i,0]$ is $< \infty$, use the maximum constant against which the clock is compared in the system, plus one (cf. [3]).

As can be seen from its definition, the computation of $HVol(z)$ is linear in the number of clocks. However, there is no need to recompute it after every operation that manipulates the zone. It only needs to be calculated as the last step of suc_τ, previous to the inclusion check. The overhead is mild, as the immediate previous operation is usually the transformation of the zone to its canonical form, which is $O(n^3)$.

The approximation is safe, as stated by Theorem 1.

Theorem 1. *If $HVol(z_1) > HVol(z_2)$ then $z_1 \not\subseteq z_2$.*

Proof. Let's assume $HVol(z_1) > HVol(z_2) \wedge z_1 \subseteq z_2$. As the operation IsIncluded() is sound and complete w.r.t. $\subseteq$, it means that $(\forall i,j)\ 0 \leq i,j \leq n$, $i \neq j \implies const(z_1[i,j]) \leq const(z_2[i,j])$. Then, as $HVol(z_1)$ and $HVol(z_2)$ are both products of the same quantity of positive terms, $HVol(z_1)$ is the product of positive numbers which are all less or equal to the corresponding ones in $HVol(z_2)$, contradicting the possibility of $HVol(z_1)$ being greater than $HVol(z_2)$.

Although hypervolume approximation resembles the convex-hull abstraction [10], there is a very important difference. Ours is an exact technique, meaning that the answer to the reachability question is responded *yes* or *no* with certainty. In convex-hull, on the other hand, zones corresponding to the same location are joined to their convex-hull over approximation. If the property is not reached, then the answer is precise, but if it is, then the answer is *maybe*.

3.2 Implementations Notes

Care must be taken when computing $HVol(z)$ as to not overflow the capacity of the container integer variable. Which type of integer variable to use for storing the $HVol$ of a zone has consequences in both memory overhead and precision. The lower the number of bits reserved for the $HVol$, the sooner it will saturate, not allowing to avoid some inclusion checks. On the other hand, if too many bits are used, the memory overhead can be considerable. The exact hypervolume would require $O(\log \Pi_{1 \leq i \leq n} |C_i|)$, where C_i is the biggest constant compared against the x_i clock in the system. As with many others time-vs-memory trade offs, experimentation should be used to find a convenient balance.

Note that the overhead depends on the implementation of DBMs. If a proper matrix is used, a `long int`, which is 8 bytes on 32 bits machines, usually provides a good amount of check saving and requires little space compared to the DBM itself. On more sophisticated representations, which leverage on Minimal Constraint Representation [15] to use a variant of linked lists of bounds, the

overhead can be variable. Our experimentation shows an average of 2% extra memory. Although Section 4 shows the experimental results, let's suppose we are dealing with a system with ten clocks (a conservative assumption). A proper DBM will have a hundred bounds. Being conservative, assume that the minimal representation has approx. 30% of the bounds. For 30 bounds and 12 bits per bound (usually enough to represent constants on the hundreds), the 360 bits can be packed in 12 `long int`. For these figures, an extra `long int` represents less than 9% of extra memory.

3.3 Sorting *Visited*

As can be seen in Fig. 1, when a new state (l, z) is found, the $Visited_{\mathsf{l}}$ set (i.e., the restriction of *Visited* [1] to states that have l as location) has to be fully explored, comparing the new zone against all the ones contained in that set.

If each zone in *Visited* has an *HVol*, then it can be turned into a priority queue, where the zones with greater *HVol* are checked first. Let z_n be the zone of the new state and z_q be the zone of $next(Visited_{\mathsf{l}})$. $HVol(z_q)$ is bigger or equal than $HVol(z)$ for every $z \in Visited_{\mathsf{l}}$. So, if $HVol(z_n)$ is greater than $HVol(z_q)$, then, because of Theorem 1, z_n is not included neither in z_q, nor in the rest of $Visited_{\mathsf{l}}$. In consequence, there is no need to continue checking, thus reducing the number of inclusion checks performed, as can be seen in the experimentation.

A problem of implementing the above mentioned technique with a traditional priority queue is that iteration is done by the successive elimination of *next* elements, thus requiring to re-insert them afterwards. For a queue with k elements, the total cost with, e.g., a heap, is $O(k \log k)$. The memory management involved in removing and adding elements can make the constants considerable, importing a noticeable overhead that can easily counter the gain from the inclusion checks avoided.

To overcome this problem, we chose a van Emde Boas tree [16], which permits nondestructive iteration. It is also convenient from a theoretical point of view: visiting the first k' elements costs $O(k' \log\log k)$. Actually van Emde Boas trees provide all of their operations in $O(\log\log k)$, at the penalty of only supporting integer numbers from a fixed interval as keys. Our experience with it was that although it can be quite difficult to implement, it provides very good performance.

The resulting algorithm is shown in Fig. 3.

Having $Visited_{\mathsf{l}}$ sorted by *HVol* is not only useful when the new zone is not included. If it is, as the bigger zones are checked first, there is a good chance that the detection occurs earlier (for instance, universal zones are always the first to be checked, whereas in a traditional implementation of $Visited_{\mathsf{l}}$ they could be "buried" deep into the set).

It should be noted that it makes no sense to check if a new state (l', z') includes one (l', z'') in *Visited* (contrary to *Pending* which is checked in line 14 of Fig. 1),

[1] Although we use a unified storage of $Visited \cup Pending$ as proposed in [4], separate indexes allows us to traverse them independently.

```
1: function IsIncludedInOrdederSet(DBM z, DBM ordered set Z)
2:     for all z' ∈ Z do
3:         if HVol(z) >HVol(z') then
4:             return NO
5:         else if IsIncluded(z, z') then
6:             return YES
7:         end if
8:     end for
9:     return NO
10: end function
```

Fig. 3. Inclusion checking algorithm (zone in ordered set of zones)

because in case it is, (l', z'') still can't be removed, as it might be part of a trace to the target state.

Section 4 shows the time and memory results for the implementation of the above mentioned techniques in the model checker Zeus. Before that, in Section 3.4, we explore the question of whether it is also worth sorting *Pending*.

3.4 Sorting *Pending*. Worth It?

At first sight the idea of sorting *Pending* by decreasing *HVol* sounds appealing: suppose that both z and z' are in *Pending*, and that $hypervolume(z) > hypervolume(z')$. As suc_τ is monotonic, it makes sense to compute $\hat{z} = suc_\tau(z)$ before $\hat{z'} = suc_\tau(z')$ because $\hat{z}$ is bigger that $\hat{z'}$. Chances are that $\hat{z'}$ might be included in $\hat{z}$, avoiding the exploration of a new zone.

An interesting aspect of sorting *Pending* that way is that it *seems* conservative. It *seems* that in case it didn't improve things, they will not get worse, i.e., no more zones will be generated.

To test these ideas, we changed the *Pending* FIFO queue into a priority queue sorted by *HVol*, and implemented it also as a van Emde Boas tree. Unfortunately, results were not positive, leading to more zones found (and thus more total time and memory) for many case studies, even ones that generated the complete state space.

The problem is that when locations are considered into the equation, things get more complicated than the intuition presented in previous paragraphs. Suppose there are (l_1, z_1) and (l_2, z_2) in *Pending* (in that order), with z_1 being $x < 10$ and z_2 being $x < 12$. Also, assume that both l_1 and l_2 have a transition to l_3 (which has a *true* invariant), but the second with an $x > 5$ guard while the first imposes no restriction.

In a FIFO exploration (l_1, z_1) will be explored first, leading to the discovery of (l_3, z_3), with z_3 being $x < \infty$. When (l_2, z_2) is expanded, the new state, with zone $z'_3 = 5 < x < \infty$, will already be included. On the other hand, if *Pending* was ordered by hypervolume, (l_2, z_2) would be explored first, leading to the discovery of (l_3, z'_3), which in turn will be put into *Pending* and explored before (l_1, z_1). When this state gets its turn, (l_3, z_3) will be generated, but the inclusion check will fail, thus creating a new zone to be explored.

Next section shows the experimental evidence that backs the claims made in Sections 3.1 and 3.3.

4 Experimental Results

To validate the proposed techniques, we incorporated them into a monoprocessor version of the ZEUS distributed model checker [17] and ran a series of experiments against well known case studies from the literature. For some of them, a reduced version (created with OBSSLICE [8], a safe model reducer) was also used and is primed in the tables. Each of them comprises two versions: the one where the property is reached (true) and the unreachable one (false). The difference is usually a changed delay. The examples used are:

1. *RCS5*, the *Railroad Crossing System* inspired in [18] with 5 trains.
2. *Pipe6*, end-to-end signal propagation in a pipe-line of sporadic processes that forward a signal emitted by a quasi periodic source, with 6 stages.
3. *FDDI8*, an extension of the FDDI token Model ring protocol where the observer monitors the time the token takes to return to a given station.
4. *Conveyor6ABC*, *Conveyor Belt* [17] (with 6 stages and 3 objects)
5. *MinePump*, a design of a fault-detection mechanism for a distributed mine-drainage controller [19].

Table 1 summarizes the sizes of the examples used in this article. The experiments were run on a Intel Pentium IV 3.0 GHz processor with 2 GB of RAM, running the Linux 2.6 operating system.

Table 1. Examples sizes

Example	Components	Clocks	Reachable locations
MinePump	10	10	1428
RCS5	8	8	1617
Conveyor6ABC	11	12	31443
FDDI8	15	23	4608

Table 2 shows the results obtained with each method independently and Table 3 the combination of both. The first columns report the time, memory and number of inclusion checks for the standard version, and then the time and number of inclusion checks for each optimization, with the percentage in parenthesis (negative values for saving, positive for increase). Memory is not reported for the first optimization because the overhead was always less than 2% (consistently with the reasoning already mentioned in Section 3.1). The overhead of the second comes from the van Emde Boas tree, which requires many pointers.

It should be noted that in the combined method, although not direct check is saved by *HVol*, many inverse ones (see line 14 of Fig. 1) can still be avoided.

Table 2. Results obtained for each method

Example	Standard			With *HVol*		*Visited* priority queue		
	Time (secs)	Mem (MB)	#checks ($\times 10^6$)	Time (secs)	#checks ($\times 10^6$)	Time (secs)	Mem (MB)	#checks ($\times 10^6$)
MinePump' true	46	7	10.5	43 (-7%)	8.9 (-15%)	43 (-7%)	7.8 (+11%)	9.2 (-12%)
MinePump' false	527	19	206.0	475 (-10%)	173.8 (-16%)	465 (-12%)	21.2 (+12%)	180.6 (-12%)
MinePump true	2807	56	1124.3	2542 (-9%)	969.7 (-14%)	2444 (-13%)	62.1 (+11%)	959.3 (-15%)
MinePump false	20580	152	4348.9	18353 (-11%)	3051.1 (-30%)	17425 (-15%)	163.2 (+7%)	3030.2 (-30%)
RCS5 true	31	7	10.2	23 (-26%)	5.5 (-46%)	28 (-10%)	7.7 (+10%)	9.1 (-11%)
RCS5 false	1039	34	341.3	952 (-8%)	286.4 (-16%)	955 (-8%)	34.6 (+2%)	311.7 (-9%)
Conveyor6ABC true	1385	183	163.6	1280 (-8%)	104.8 (-36%)	1338 (-4%)	198.0 (+8%)	138.5 (-15%)
Conveyor6ABC false	4142	280	998.8	3368 (-19%)	567.1 (-43%)	3600 (-13%)	306.4 (+9%)	762.7 (-23%)
FDDI8 true	488	20	0.1	488 (0%)	0.1 (0%)	486 (-<1%)[2]	21.0 (+5%)	0.1 (-<1%)
FDDI8 false	7642	120	18.2	7645 (+<1%)	18.2 (0%)	7629 (-<1%)	122.0 (+2%)	18.2 (-<1%)

Table 3. Results obtained with both methods combined

Example	Both methods		
	Time (secs)	Mem (MB)	#checks ($\times 10^6$)
MinePump' true	43 (-7%)	7.8 (+11%)	8.3 (-21%)
MinePump' false	465 (-12%)	21.2 (+12%)	181.3 (-12%)
MinePump true	2436 (-13%)	62.1 (+11%)	955.7 (-15%)
MinePump false	17388 (-16%)	163.2 (+7%)	2348.4 (-46%)
RCS5 true	23 (-26%)	7.5 (+10%)	8.0 (-22%)
RCS5 false	926 (-11%)	34.6 (+2%)	286.7 (-16%)
Conveyor6ABC true	1295 (-6%)	198.0 (+8%)	115.9 (-29%)
Conveyor6ABC false	3470 (-16%)	306.4 (+9%)	569.3 (-43%)
FDDI8 true	487 (-<1%)	21.0 (+5%)	0.1 (-<1%)
FDDI8 false	7652 (+<1%)	122.0 (+2%)	18.2 (-<1%)

As can be seen in Table 2, *HVol* features time savings of up to 19% (not counting *RCS5* true, because it already takes a very short time and is only presented for completeness), and sorting *Visited* of up to 15%. The first one has neglectable memory overhead, while the second uses around 10-12%. There is no

[2] Marginal improvements, not seen in figures because of rounding.

direct relationship between the number of checks saved and the speedup. This is because each case studies has different sized matrices and for each check saved, the number of cell matrices to be compared differs.

FDDI8 is an interesting case study because, as no inclusion check could be avoided with *HVol*, the difference in times measures pure overhead, showing that the method is very light.

Table 2 also shows that different case studies benefit the most from different techniques. If memory is a premium, clearly *HVol* is convenient, but if some memory can be spent in order to obtain earlier results, then a decision should be made among the two. Also, they can be combined. As shown in Table 3, with the exception of *FDDI8*, the combined method is never worse than the worse of the optimizations (see for instance *Conveyor6ABC*). Sometimes it is as good as the best of them (*MinePump'*, *MinePump* true, *RCS5* true) and sometimes better (*MinePump* false, *RCS5* false).

Both the second and the combined method have indeed a memory overhead that seems, in terms of percentages, on the same order of magnitude as the time saved. However, in cases like *MinePump* false on Table 3, it can be seen that the 16% saving of time translates to almost one hour of almost six, at the cost of 7% more memory, which only amounts to less than 12 extra MB of RAM.

5 Conclusions and Future Work

In this article we presented two techniques that contribute to speed up forward reachability. They are based on approximating the hypervolume of the polyhedra that represents the valuations of clocks in timed automata model checking. Although the hypervolume is approximated, both techniques give exact answers.

The first is based on avoiding inclusion checks when the approximate hypervolumes makes the inclusion impossible. The other, in sorting the already-visited-states set according to the approximate hypervolumes, avoiding to traverse some parts of it while checking for included zones. The second is only possible thanks to a very optimized implementation of a van Emde Boas tree [16], but the first is quite simple.

These techniques can be used independently and obtain interesting speedups. For instance, in cases like *MinePump* false in Table 3 it can be seen that the 16% saving of time translates to almost one hour of almost six, at the cost of 7% more memory, which only amounts to less than 12 extra MB of RAM. According to our experiments, some models benefit more with one of them, and some with the others, with acceleration of up to 19 and 15% respectively. The first one has neglectable memory overhead, the second, a moderate one (10-12%).

They can also be combined, although the speedups are not additive, because there is some mutual cancellation. The good news is that there is no need to speculate on which one to use, because using both is generally as good as using the best of them.

It should be noted that the techniques are very unobtrusive, in the sense that they are orthogonal to many other optimizations such as [3,4,5,6,8,9], allowing to use all of them together.

Also, we showed that applying the same ideas to the *Pending* queue –where the states that remain unexplored are kept– can have negative impact. In order to reverse that, *Pending* should be separated by location, and then sorted, but that will increase the cost of adding newly discovered zones to it. Experimentation with different data structures towards that end is a yet-to-be-explored area.

Although we chose a van Emde Boas tree to sort the *Visited* queue, some less modular yet simpler implementations –based on linked lists of states, with pointers marking insertion places– are possible. This trade-off should be revisited to see if some of the overhead can be avoided.

To pursue further in this line of research, it would be interesting to analyze which topological characteristics of the model influence in each method's performance. A consequence of this could be an on-the-fly detection method, that switches between them.

Acknowledgements. We would like to thanks the insightful comments of the anonymous referees that helped us improve the paper.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Bouajjani, A., Tripakis, S., Yovine, S.: On-the-Fly Symbolic Model-Checking for Real-Time Systems. In: 18^{th} IEEE Real-Time Systems Symposium (RTSS '97), San Francisco, USA, IEEE Computer Scienty Press, Los Alamitos (1997)
3. Behrmann, G., Bouyer, P., Larsen, K., Pelnek, R.: Lower and upper bounds in zone based abstractions of timed automata. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 312–326. Springer, Heidelberg (2004)
4. Behrmann, G., David, A., Larsen, K.G., Yi, W.: Unification & sharing in timed automata verification. In: Ball, T., Rajamani, S.K. (eds.) Model Checking Software. LNCS, vol. 2648, pp. 225–229. Springer, Heidelberg (2003)
5. Bengtsson, J.: Reducing memory usage in symbolic state-space exploration for timed systems. Technical Report 2001-009, Department of Information Technology, Uppsala University (2001)
6. Daws, C., Yovine, S.: Reducing the number of clock variables of timed automata. In: Proceedings IEEE Real-Time Systems Symposium (RTSS '96), pp. 73–81. IEEE Computer Society Press, Los Alamitos (1996)
7. Wang, F.: Efficient verification of timed automata with BDD-like data-structures. In: Zuck, L.D., Attie, P.C., Cortesi, A., Mukhopadhyay, S. (eds.) VMCAI 2003. LNCS, vol. 2575, pp. 189–205. Springer, Heidelberg (2002)
8. Braberman, V., Garbervetsky, D., Olivero, A.: ObsSlice: A timed automata slicer based on observers. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, Springer, Heidelberg (2004)
9. Braberman, V., Olivero, A., Schapachnik, F.: Optimizing timed automata model checking via clock reordering. In: 27^{th} IEEE International Real-Time Systems Symposium, Work in Progress Session, IEEE, Los Alamitos (2006)

10. Balarin, F.: Approximate reachability analysis of timed automata. In: Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96), p. 52. IEEE Computer Society, Washington, DC, USA (1996)
11. 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)
12. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: UPPAAL - a tool suite for automatic verification of real-time systems. In: Hybrid Systems, pp. 232–243. Springer, Heidelberg (1995)
13. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) Automatic Verification Methods for Finite State Systems. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
14. Yovine, S.: Model checking timed automata. In: Rozenberg, G. (ed.) Lectures on Embedded Systems. LNCS, vol. 1494, Springer, Heidelberg (1998)
15. Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Compact data structures and state-space reduction for model-checking real-time systems. Real-Time Syst. 25(2-3), 255–275 (2003)
16. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Mathematical Systems Theory 10, 99–127 (1977)
17. Braberman, V., Olivero, A., Schapachnik, F.: Issues in Distributed Model-Checking of Timed Automata: building ZEUS. International Journal of Software Tools for Technology Transfer 7, 4–18 (2005)
18. Alur, R., Courcoubetis, C., Dill, D., Halbwachs, N., Wong-Toi, H.: An implementation of three algorithms for timing verification based on automata emptiness. In: Proceedings of the 13th IEEE Real-time Systems Symposium, Phoenix, Arizona, pp. 157–166 (1992)
19. Braberman, V.: Modeling and Checking Real-Time Systems Designs. Ph d. thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires (2000)

Counter-Free Input-Determined Timed Automata*

Fabrice Chevalier[1], Deepak D'Souza[2], and Pavithra Prabhakar[3]

[1] Laboratoire Spécification et Vérification,
Ecole Normale Superieure de Cachan, France
fabrice.chevalier@lsv.ens-cachan.fr
[2] Dept. of Computer Science & Automation
Indian Institute of Science, Bangalore 560012, India.
deepakd@csa.iisc.ernet.in
[3] Dept. of Computer Science
University of Illinois at Urbana-Champaign, USA.
pprabha2@uiuc.edu

Abstract. We identify a class of timed automata, which we call counter-free input-determined automata, which characterize the class of timed languages definable by several timed temporal logics in the literature, including MTL. We make use of this characterization to show that MTL+Past satisfies an "ultimate stability" property with respect to periodic sequences of timed words. Our results hold for both the pointwise and continuous semantics. Along the way we generalize the result of McNaughton-Papert to show a counter-free automata characterization of FO-definable finitely varying functions.

1 Introduction

A number of classes of timed automata based on "input-determined" distance operators have been proposed in the literature. These include the event-recording automata of [AFH94, HRS98], which make use of the operators $\triangleleft_a$ and $\triangleright_a$ (which measure the distance to the last and next a's respectively), state-clock automata [RS99], and eventual timed automata [DM05, CDP06], which make use of the "eventual operator" $\Diamond_a$ inspired by Metric Temporal Logic (MTL) [Koy90, AFH96, OW05]. In [DT04, CDP06] these operators were abstracted into a general notion of an input-determined operator, and the corresponding classes of timed automata called input-determined automata or IDA's (parameterized by a set of input-determined operators) were shown to have robust logical properties, including a monadic second-order logic characterization, and expressively complete (with respect to the first-order fragment of the corresponding MSO logics) timed temporal logics based on these operators. However, an important link that remained unexplored was a characterization of the class of automata which correspond to these temporal logics, along the lines of the classical characterization via counter-free automata for discrete temporal logic, which follows from the work of Kamp and McNaughton-Papert [Kam68, MP71].

In this paper our aim is to fill this gap. We identify a class of counter-free IDA's (again parameterized by a set of input-determined operators) that precisely characterize

* This work was partly supported by P2R Timed-DISCOVERI.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 82–97, 2007.

the class of timed languages definable by the timed temporal logics based on these operators. Our class of counter-free IDA's comprise "proper" IDA's (which are in a sense a determinized form of IDA's) whose underlying graphs have no counters in the classical sense. Our results hold for both the "pointwise" and "continuous" semantics for timed formalisms.

For the pointwise semantics, we can simply factor through the classical results of Kamp and McNaughton-Papert, and the translations set up in [DT04]. For the continuous case we first prove an analogue of the McNaughton-Papert result for finitely varying functions, by characterizing the first-order definable languages of finitely-varying functions in terms of a counter-free fragment of a class of automata we call ST-NFA's. Once we have this result, we can essentially factor through the translations for continuous time set up in [CDP06].

We emphasize that this general result gives us automata characterizations for several of the timed logics proposed in the literature, including EventClockTL [HRS98], Metric Interval Temporal Logic (MITL) [AFH96], and MTL [AFH96, OW05], for both the pointwise and continuous semantics. Among other applications, such characterizations can be useful in arguing expressiveness results for these logics.

In the final part of this paper we make use of this characterization to prove a property of timed languages definable by MTL^{c}+Past, namely that they must satisfy an "ultimate stability" property with respect to a periodic sequence of timed words. Thus, given a periodic sequence of finite timed words of the form uv^iw, the truth of an MTL^{c}+Past formula at any real time point, must eventually stabilize (i.e. become always true, or always false) along the models in the sequence. This is a stronger result than a property for MTL proved in [PD06], in that it holds for MTL with past operators, and furthermore does not depend on the "duration" of the timed word v in the periodic sequence.

In the sequel we concentrate on the continuous semantics. The details for the pointwise semantics and other arguments not included here for space reasons, can be found in the technical report [CDP07].

2 Preliminaries

For an alphabet A, we use A^* to denote the set of finite words over A. For a word w in A^*, we use $|w|$ to denote its length. The set of non-negative reals and rationals will be denoted by $\mathbb{R}_{\geq 0}$ and $\mathbb{Q}_{\geq 0}$ respectively. We will deal with intervals of non-negative reals, i.e. convex subsets of $\mathbb{R}_{\geq 0}$, and denote by $\mathcal{I}_{\mathbb{R}_{\geq 0}}$ and $\mathcal{I}_{\mathbb{Q}_{\geq 0}}$ the set of such intervals with end-points in $\mathbb{R}_{\geq 0} \cup \{\infty\}$ and $\mathbb{Q}_{\geq 0} \cup \{\infty\}$ respectively. Two intervals I and J will be called *adjacent* if $I \cap J = \emptyset$ and $I \cup J$ is an interval.

Let A be an alphabet and let $f : [0, r] \to A$ be a function, where $r \in \mathbb{R}_{\geq 0}$. We use $dur(f)$ to denote the duration of f, which in this case is r. A point $t \in (0, r)$ is a point of *continuity* of f if there exists $\epsilon > 0$ such that f is constant in the interval $(t-\epsilon, t+\epsilon)$. All other points in $[0, r]$ are points of *discontinuity* of f. We say f is *finitely varying* if it has only a finite number of discontinuities. We denote by $FVF(A)$ the set of all finitely varying functions over A.

An *interval representation* for a finitely varying function $f : [0, r] \to A$ is a sequence of the form $(a_0, I_0) \cdots (a_n, I_n)$, with $a_i \in A$ and $I_i \in \mathbb{I}_{\mathbb{R}_{\geq 0}}$, satisfying the conditions

that the union of the intervals is $[0, r]$, each I_i and I_{i+1} are adjacent, and for each i, f is constant and equal to a_i in the interval I_i. We can obtain a *canonical* interval representation for f by putting each point of discontinuity in a singular interval by itself. Thus the above interval representation for f is canonical if n is even, for each even i I_i is singular (i.e. of the form $[t, t]$), and for no even i such that $0 < i < n$ is $a_{i-1} = a_i = a_{i+1}$.

A canonical interval representation for a function gives us a canonical way of "untiming" the function: thus if $(a_0, I_0) \cdots (a_{2n}, I_{2n})$ is the canonical interval representation for a function f, then we define $untiming(f)$ to be the string $a_0 \cdots a_{2n}$ in A^*. The untiming thus captures explicitly the value of the function at its points of discontinuity and the open intervals between them. Note that strings which represent the untiming of a function will always be of odd length and for no even position i will the letters at positions $i-1$, i, and $i+1$ be the same. We call such words *canonical*. A canonical word w can be "timed" to get a function in a natural way: thus a function f is in $timing(w)$ if $untiming(f) = w$. We extend the definition of $timing$ and $untiming$ to languages of functions and words in the expected way.

We now turn to some notions regarding classical automata and a variant we introduce. Recall that a non-deterministic finite state automaton (NFA) over an alphabet A is a structure $\mathcal{A} = (Q, s, \delta, F)$, where Q is a finite set of states, s is the initial state, $\delta \subseteq Q \times A \times Q$ is the transition relation, and $F \subseteq Q$ is the set of final states. A run of $\mathcal{A}$ on a word $w = a_0 \cdots a_n \in A^*$ is a sequence of states $q_0, \ldots, q_{n+1}$ such that $q_0 = s$, and $(q_i, a_i, q_{i+1}) \in \delta$ for each $i \leq n$. The run is accepting if $q_{n+1} \in F$. The symbolic language accepted by $\mathcal{A}$, denoted $L_{sym}(\mathcal{A})$, is the set of words in A^* over which $\mathcal{A}$ has an accepting run. Languages accepted by NFA's are called *regular* languages. We say the NFA $\mathcal{A}$ is *deterministic* (and call it a DFA) if the transition relation δ is a function from $Q \times A$ to Q. A well-known fact is that every regular language L has a unique (up to isomorphism) minimal state DFA accepting it, which we refer to as $\mathcal{A}_L$.

A *counter* in an NFA $\mathcal{A}$ is a sequence of distinct states $q_0, \ldots, q_n$ with $n \geq 1$, along with a word $u \in A^*$, such that there is a path labeled u in $\mathcal{A}$ from q_i to q_{i+1} (for each $i \in \{0, \ldots, n-1\}$) and from q_n to q_0. An NFA is said to be *counter-free* if it does not contain a counter. A regular language is said to be *counter-free* if there exists a counter-free NFA for it. We will call a sequence of words in A^* $\langle w_i \rangle = w_0, w_1, \cdots$ *periodic* if there exist strings u, v, w in A^* such that $w_i = uv^i w$ for each i. We say a language $L \subseteq A^*$ is *ultimately stable* (with respect to periodic sequences of words) if for each periodic sequence $\langle w_i \rangle$ there exists a $k \geq 0$, such that for all $i \geq k$, $w_i \in L$ or for all $i \geq k$, $w_i \notin L$.

Proposition 1. *Let L be a regular language over an alphabet A. Then the following are equivalent: (1) L is counter-free, (2) $\mathcal{A}_L$ is counter-free, (3) L is ultimately stable with respect to periodic sequences of words.* □

Using the above proposition, it follows that counter-free regular languages are closed under the boolean operations of union, intersection and complement.

We now define a variant of NFA's called *state-transition-labeled* NFA*'s* or ST-NFA$'s$ for short, which are convenient for generating finitely varying functions. An ST-NFA over A is a structure $\mathcal{A} = (Q, s, \delta, F, l)$ similar to an NFA over A, except that $l : Q \longrightarrow A$ labels states with letters from A. The ST-NFA $\mathcal{A}$ accepts strings of

the form $A(AA)^*$. A run of $\mathcal{A}$ on a string $w = a_0a_1 \cdots a_{2n}$ in $A(AA)^*$, is a sequence of states $q_0, \ldots, q_{n+1}$ satisfying $q_0 = s$, $(q_i, a_{2i}, q_{i+1}) \in \delta$ for $i \in \{0, \cdots, n\}$ and $l(q_i) = a_{2i-1}$ for each $i \in \{1, \ldots, n\}$; it is accepting if $q_{n+1} \in F$. We define $L_{sym}(\mathcal{A})$ to be the set of strings $w \in A^*$ on which $\mathcal{A}$ has an accepting run.

An ST-NFA $\mathcal{A}$ also generates functions in a natural way: we begin by taking a transition emanating from the start state, emitting its label, and then spend time at the resulting state emitting its label all the while, before taking a transition again; and so on. The language of finitely-varying functions defined by an ST-NFA $\mathcal{A}$ is defined to be $timing(L_{sym}(\mathcal{A}))$, and noted $F(\mathcal{A})$. For convenience we will stick to *canonical* ST-NFA's which are ST-NFA$'s$ in which we never have an a-labelled transition between an a-labelled source and target state, for any $a \in A$. It is not difficult to see that any ST-NFA can be converted to a canonical one whose function language is the same. We note that for a canonical ST-NFA $\mathcal{A}$, $untiming(F(\mathcal{A})) = L_{sym}(\mathcal{A})$.

We now define the counter-free version of ST-NFA's. A counter in an ST-NFA is similar to one in an NFA, except that by the "label" of a path in the automaton we mean the sequence of alternating state and transition labels along the path. Thus the label of the path $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots q_n \xrightarrow{a_n} q_{n+1}$ is $l(q_0)a_0l(q_1)a_1 \cdots l(q_n)a_n$. We note that a counter-free ST-NFA can define a language (e.g. the single state, single transition ST-NFA over $\{a\}$ which defines the language $a(aa)^*$) which is *not* counter-free in the classical sense. However, if we consider ST-NFA's over a *partitioned* alphabet, where the alphabet A is partitioned into A_1 and A_2 which label transitions and states respectively, then we can show that:

Proposition 2. *Let (A_1, A_2) be a partitioned alphabet. A regular subset of $A_1(A_2A_1)^*$ is counter-free iff there exists a counter-free* ST-NFA *over (A_1, A_2) accepting it.* □

Finally, by going over to an alternating alphabet (say by renaming transition labels) using the closure properties of classical and counter-free languages, and then coming back, we can verify that:

Proposition 3. *Each of the classes of function languages definable by* ST-NFA*'s and counter-free* ST-NFA*'s over an alphabet A are closed under boolean operations.* □

3 Counter-Free Continuous Input-Determined Automata

We begin with some notation. We define a timed word σ over an alphabet Σ to be an element of $(\Sigma \times \mathbb{R}_{\geq 0})^*$, such that $\sigma = (a_0, t_0)(a_1, t_1) \cdots (a_n, t_n)$ and $t_0 < t_1 < \cdots < t_n$. We write $dur(\sigma)$ to denote the duration of σ, i.e. t_n above. We denote the set of timed words over Σ to be $T\Sigma^*$. Given timed words $\sigma = (a_0, t_0) \cdots (a_n, t_n)$ and $\sigma' = (a'_0, t'_0) \cdots (a'_k, t'_k)$ with $t'_0 > 0$, we define their concatenation $\sigma \cdot \sigma'$ in the standard way to be $(a_0, t_0) \cdots (a_n, t_n)(a'_0, t_n + t'_0) \cdots (a'_k, t_n + t'_k)$.

We define an *input-determined operator* Δ over an alphabet Σ as a partial function from $(T\Sigma^* \times \mathbb{R}_{\geq 0})$ to $2^{\mathbb{R}_{\geq 0}}$, which is defined for all pairs (σ, t), where $t \in [0, dur(\sigma)]$. Thus an input-determined operator identifies a set of "distances" for a given timed word and a time point in it. Given a set of input-determined operators Op, we define the set of guards over Op, denoted by $\mathcal{G}(Op)$, inductively as $g ::= \top \mid \Delta^I \mid \neg g \mid g \vee g \mid g \wedge g$, where

$\Delta \in Op$ and $I \in \mathcal{I}_{\mathbb{Q}_{\geq 0}}$. Given a timed word σ, we define the satisfiability of a guard g at time $t \in [0, dur(\sigma)]$, denoted $\sigma, t \models g$, as follows: $\sigma, t \models \Delta^I$ iff $\Delta(\sigma, t) \cap I \neq \emptyset$, with boolean operators treated in the expected way.

For example, the operator $\Delta_{\mathbb{Q}}$ which maps (σ, t) to $\{1\}$ if t is rational and to $\{0\}$ otherwise, is an input-determined operator. Other examples include the eventual operator $\diamond_a$, inspired by MTL, which maps (σ, t) to the set of distances d such that an a occurs at time $t + d$ in σ; and the event-recording operator $\triangleleft_a$ which maps (σ, t) to the (empty or singleton) set of distances to the last occurrence of the event a before time t.

We call an input-determined operator Δ over Σ *finitely varying* if for each $I \in \mathcal{I}_{\mathbb{Q}_{\geq 0}}$, and each $\sigma \in T\Sigma^*$, the characteristic function $f^{\sigma}_{\Delta^I} : [0, dur(\sigma)] \to \{0, 1\}$ of Δ^I, defined as $f^{\sigma}_{\Delta^I}(t)$ is 1 if $\sigma, t \models \Delta^I$, and 0 otherwise, is finitely varying. Of the example operators above, $\diamond_a$ and $\triangleleft_a$ are finitely varying, while $\Delta_{\mathbb{Q}}$ is not.

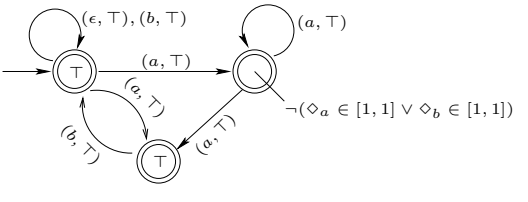
Let Σ be an alphabet and Op be a set of input determined operators over Σ. We call (Γ_1, Γ_2) a *symbolic alphabet* over (Σ, Op), if Γ_1 is a finite subset of $(\Sigma \cup \{\epsilon\}) \times \mathcal{G}(Op)$ and Γ_2 is a finite subset of $\mathcal{G}(Op)$. We define the set of timed words over Σ associated with a function f in $FVF(\Gamma_1 \cup \Gamma_2)$, denoted $tw(f)$, as follows. If $untiming(f) \notin \Gamma_1(\Gamma_2\Gamma_1)^*$, then $tw(f) = \emptyset$. Otherwise, a timed word $\sigma = (a_0, t_0) \cdots (a_n, t_n)$ is in $tw(f)$, provided for all $t \in [0, dur(f)]$,

- If $f(t) = (a, g)$, for some $a \in \Sigma$ and $g \in \mathcal{G}(Op)$, then $\sigma, t \models g$, and there exists i in $\{0, \cdots, n\}$, with $t_i = t$ and $a_i = a$.
- If $f(t) = (\epsilon, g)$ or g, for some $g \in \mathcal{G}(Op)$, then $\sigma, t \models g$, and there does not exist i in $\{0, \cdots, n\}$ with $t_i = t$.

Note that for any f, $tw(f)$ is either empty or a singleton set. We extend the definition of tw to sets of functions, as the union of the timed words corresponding to each function in the set.

Let Σ be an alphabet and Op be a set of input-determined operators based on Σ. A *Continuous Input Determined Automaton* (*CIDA*) $\mathcal{A}$ over (Σ, Op) is simply an ST-NFA over a symbolic alphabet (Γ_1, Γ_2) based on (Σ, Op). As an ST-NFA $\mathcal{A}$ defines a language of functions $F(\mathcal{A})$. We are however more interested in the timed language it accepts, denoted $L(\mathcal{A})$, and defined to be $tw(F(\mathcal{A}))$.

Here is a concrete example of a *CIDA* over the set of "eventual operators" $Op = \{\diamond_a \mid a \in \Sigma\}$. The diagram shows a *CIDA* over $(\{a, b\}, Op)$ which recognizes the language L_{ni} (for "no insertions"), which consists of timed words in which between any two consecutive a's, there is no time point from which there is an a or a b at a distance of one time unit in the future.



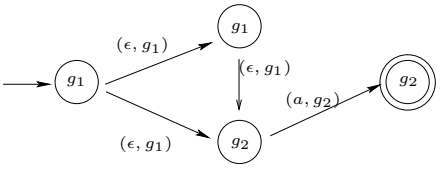
We now define *proper CIDA*'s which are a time-deterministic form of *CIDA*'s, and which we will use to define our counter-free *CIDA*'s. Let G be a finite set of atomic guards over Op. We call (Γ_1, Γ_2) the *proper* symbolic alphabet over (Σ, Op) based

on G, if $\Gamma_1 = (\Sigma \cup \{\epsilon\}) \times 2^G$ and $\Gamma_2 = 2^G$. We interpret $h \subseteq G$ as a guard which specifies precisely the guards in G that are true. Thus h is interpreted as the guard $\bigwedge_{g \in h} g \wedge \bigwedge_{g \in G-h} \neg g$.

We define a *proper CIDA* over (Σ, Op) to be an ST-NFA over a proper symbolic alphabet based on (Σ, Op). The symbolic, function, and timed languages defined by a proper *CIDA* are defined similarly to *CIDA*'s. We call a word γ over a symbolic alphabet (Γ_1, Γ_2) *fully canonical*, if $\gamma \in \Gamma_1(\Gamma_2\Gamma_1)^*$ and no subword of γ is of the form $g \cdot (\epsilon, g) \cdot g$. We call a proper *CIDA fully canonical* if its symbolic language consists of fully canonical proper words. The class of languages defined by *CIDA*'s and fully canonical proper *CIDA*'s coincide:

Lemma 1 ([CDP06]). *CIDA's over (Σ, Op) and fully canonical proper CIDA's over (Σ, Op) define the same class of timed languages.* □

The class of counter-free *CIDA*'s we are interested in this paper is the class of counter-free *CIDA*'s over (Σ, Op), is the class of fully canonical proper *CIDA*'s over (Σ, Op) whose underlying ST-NFA is counter-free. We denote this class by $CFCIDA(\Sigma, Op)$. As an example, let $\Sigma = \{a\}$, $Op = \{\diamond_a\}$ and $G = \{\diamond_a^{[1,1]}\}$. The *CFCIDA* over (Σ, Op) alongside recognizes timed words comprising exactly one a, which occurs in the interval $[1, 2]$. In the diagram, $g_1 = \{\diamond_a^{[1,2]}\}$ and $g_2 = \emptyset$.



4 Counter-Free ST-NFA's and FO-Definable Functions

In this section we show that over a partitioned alphabet (A_1, A_2), the class of first-order definable languages of finitely-varying functions (for a natural FO logic we will introduce soon) is precisely the class of function languages defined by counter-free ST-NFA's over (A_1, A_2).

For an alphabet A, the syntax of the first order logic $\mathsf{FO}^{\mathsf{c}}(A)$, is given by:

$$\varphi ::= Q_a(x) \mid x < y \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \exists x\varphi,$$

where $a \in A$, and x and y are variables.

We interpret a formula φ of the logic over a finitely varying function f in $FVF(A)$, along with an interpretation $\mathbb{I}$ with respect to f, which assigns to each variable a value in $[0, dur(f)]$. For an interpretation $\mathbb{I}$, we use the notation $\mathbb{I}[t/x]$ to denote the interpretation which sends x to t and agrees with $\mathbb{I}$ on all other variables. Given a formula $\varphi \in \mathsf{FO}^{\mathsf{c}}(A)$, $f \in FVF(A)$, and an interpretation $\mathbb{I}$ with respect to f to the variables in φ, the satisfaction relation $f, \mathbb{I} \models \varphi$, is defined inductively (with boolean operators handled in the usual way) as:

$$\begin{array}{l} f, \mathbb{I} \models \mathrm{Q}_a(x) \text{ iff } f(\mathbb{I}(x)) = a, \text{ where } a \in A. \\ f, \mathbb{I} \models x < y \text{ iff } \mathbb{I}(x) < \mathbb{I}(y). \\ f, \mathbb{I} \models \exists x\varphi \quad \text{iff } \exists t \in [0, dur(f)] : f, \mathbb{I}[t/x] \models \varphi. \end{array}$$

For a sentence φ (a formula without free variables) in $\mathsf{FO}^{\mathsf{c}}(A)$, the interpretation does not play any role, and we set the language of functions defined by φ to be $F(\varphi) = \{f \in FVF(A) \mid f \models \varphi\}$.

As an example, the formula $\varphi_{cont} = \exists y \exists z(y < x \wedge x < z \wedge \bigvee_{a \in A} \forall u(y < u \wedge u < z \Rightarrow Q_a(u) \wedge Q_a(x)))$ asserts that the point x is a point of continuity. As another example, for the partitioned symbolic alphabet (Γ_1, Γ_2) based on some (Σ, Op), the $\mathsf{FO}^{\mathsf{c}}(\Gamma_1 \cup \Gamma_2)$ formula $\varphi_{fc} = \forall x(\varphi_{disc}(x) \Rightarrow \neg(\bigvee_{(\epsilon,g) \in \Gamma_1} Q_{(\epsilon,g)}(x) \wedge \exists y \exists z(y < x \wedge x < z \wedge \forall u(u \neq x \wedge y < u \wedge u < z \Rightarrow Q_g(u)))))$, (where $\varphi_{disc} = \neg\varphi_{cont}$) asserts that the untiming of the function is fully canonical.

For a partitioned alphabet (A_1, A_2) we call a finitely varying function f in $FVF(A_1 \cup A_2)$ *alternating* if $untiming(f) \in A_1(A_2A_1)^*$ (thus the discontinuities are labelled by symbols in A_1 and continuities by labels in A_2). Let $alt\text{-}FVF(A_1, A_2)$ denote the class of alternating finitely varying functions over (A_1, A_2).

Theorem 1. *Let (A_1, A_2) be a partitioned alphabet with $A = A_1 \cup A_2$. Then the class of $\mathsf{FO}^{\mathsf{c}}(A)$-definable languages of alternating finitely-varying functions over (A_1, A_2) is precisely the class of function languages definable by counter-free* ST-NFA*'s over (A_1, A_2).*

The rest of this section is devoted to a proof of this theorem. We recall briefly the logic LTL and its two interpretations, one over discrete words and the other over functions. The syntax of $\mathsf{LTL}(A)$ is given by:

$$\theta ::= a \mid (\theta U \theta) \mid (\theta S \theta) \mid \neg\theta \mid (\theta \vee \theta),$$

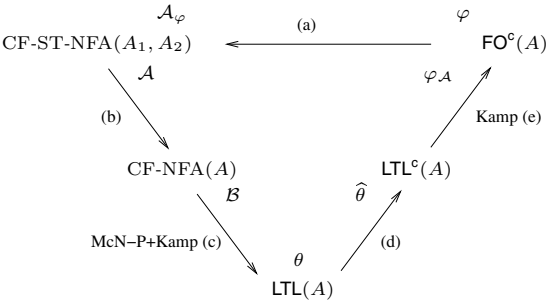
where $a \in A$. The logic is interpreted over words in A^*, with the following semantics. Given a word $w = a_0 \cdots a_n$ in A^* and a position $i \in \{0, \ldots, n\}$, we say $w, i \models a$ iff $a_i = a$; and $w, i \models \theta U \eta$ iff there exists j such that $i < j \leq n$, $w, j \models \eta$ and for all k such that $i < k < j$, $w, k \models \theta$. The "since" operator S is defined in a symmetric way to U in the past, and the boolean operators in the usual way. We denote by $L_{sym}(\theta)$ the set $\{w \in A^* \mid w, 0 \models \theta\}$.

The logic LTL can also be interpreted over functions as done in [Kam68]. Here we restrict the models to finitely-varying functions in $FVF(A)$, and we denote this logic by $\mathsf{LTL}^{\mathsf{c}}(A)$. Given a function $f \in FVF(A)$, $t \in [0, dur(f)]$ and $\theta \in \mathsf{LTL}^{\mathsf{c}}(A)$, the satisfaction relation $f, t \models \theta$ is defined as follows:

$$\begin{array}{ll} f, t \models a & \text{iff } f(t) = a. \\ f, t \models \theta U \eta & \text{iff } \exists t' : t < t' \leq dur(f),\ f, t' \models \eta, \text{ and } \forall t'' : t < t'' < t', f, t'' \models \theta. \\ f, t \models \theta S \eta & \text{iff } \exists t' : 0 \leq t' < t,\ f, t' \models \eta, \forall t'' : t' < t'' < t,\ f, t'' \models \theta. \end{array}$$

The boolean operators are interpreted in the expected way. We set $F(\theta) = \{f \in FVF(A) \mid f, 0 \models \theta\}$. As an example, the $\mathsf{LTL}^{\mathsf{c}}(A)$ formulas $\theta_{cont} = \bigvee_{a \in A}(a \wedge (aSa) \wedge (aUa))$ and $\theta_{disc} = \neg\theta_{cont}$ characterize the points of continuity and discontinuity respectively in a function over A.

Returning now to the proof of Theorem 1, the route we follow is given schematically in the figure below.



Step (a): Let φ be a sentence in $\mathsf{FO}^c(A)$. We show how to construct a counter-free ST-NFA $\mathcal{A}_\varphi$ over A, such that $F(\mathcal{A}_\varphi) = F(\varphi)$.

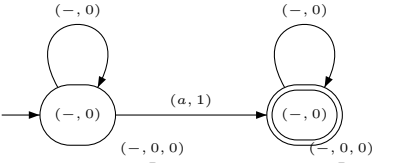
The proof proceeds in a similar manner to the one in [CDP06]. We will represent models of formulas with free variables in them, as functions with the interpretations built into them. We assume an ordering on the countable set of first-order variables given by $x_1, x_2, \cdots$. For a formula φ with free variables among $X = \{x_{i_1}, \cdots, x_{i_m}\}$ (in order), we represent a function f and an interpretation $\mathbb{I}$ as a function $f^X_\mathbb{I} : [0, dur(f)] \to A \times \{0,1\}^m$ given by $f^X_\mathbb{I}(t) = (f(t), b_1, \cdots, b_m)$, where $b_k = 1$ iff $\mathbb{I}(x_{i_k}) = t$. Thus for a formula φ with free variables in X we have a notion of X-models of φ.

Proposition 4. *Let φ be an $\mathsf{FO}^c(A)$ formula with free variables X and let $\mathcal{A}$ be a counter-free* ST-NFA *accepting the X-models of φ. Then for any set of variables X' which contains X, we can construct a counter-free* ST-NFA *$\mathcal{A}'$ accepting precisely the X'-models of φ.* □

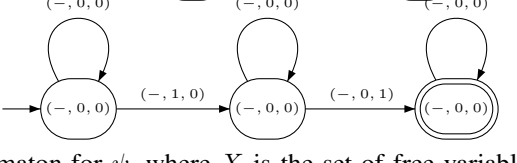
Lemma 2. *Let φ be an $\mathsf{FO}^c(A)$ formula and let X be the set of free variables in it. Then we can construct a counter-free* ST-NFA *$\mathcal{A}^X_\varphi$ which accepts precisely the X-models of φ.*

Proof. The idea of the proof is similar to the one in [CDP06], except that now we need to also ensure that the automaton we obtain is counter-free. For a set of variables Y, let $\mathcal{A}^Y_{valid}$ denote the ST-NFA which accepts all "valid" Y-models. It is easy to construct this ST-NFA and to check that it is counter-free. We construct the counter-free ST-NFA $\mathcal{A}^X_\varphi$ by induction on the structure of φ.

1. $\varphi = Q_a(x)$: The automaton $\mathcal{A}^{\{x\}}_\varphi$ is:



2. $\varphi = x < y$: The automaton $\mathcal{A}^{\{x,y\}}_\varphi$ (assuming x occurs before y in the variable ordering) is:



3. $\varphi = \neg\psi$: Let $\mathcal{A}^X_\psi$ be the automaton for ψ, where X is the set of free variables in ψ. Then $\mathcal{A}^X_\varphi$ is the intersection of $\mathcal{A}^X_{valid}$ with the counter-free ST-NFA that recognizes the complement of the function language of $\mathcal{A}^X_\psi$ (cf. Prop 3).

4. $\varphi = \psi \vee \nu$: Let $\mathcal{A}_\psi^{X'}$ be the counter-free ST-NFA for ψ, where X' is the set of free variables in ψ, and let $\mathcal{A}_\nu^{X''}$ be the counter-free ST-NFA for ν, where X'' is the set of free variables in ν. Let $X = X' \cup X''$. By Prop. 4 we obtain ST-NFA's $\mathcal{A}_\psi^X$ and $\mathcal{A}_\nu^X$. Then $\mathcal{A}_\varphi^X$ is the union of $\mathcal{A}_\psi^X$ and $\mathcal{A}_\nu^X$.
5. $\varphi = \exists x \psi$: Let X' be the set of free variables in ψ so that $X = X' - \{x\}$. Let $\mathcal{A}_\psi^{X'}$ be a counter-free ST-NFA for ψ. Without loss of generality we can assume $\mathcal{A}_\psi^{X'}$ has no "useless" states (i.e. those which cannot be reached from the start state or cannot reach a final state). Now we simply project away the component corresponding to x in the symbols on the transitions of $\mathcal{A}_\psi^{X'}$ to obtain the required counter-free ST-NFA $\mathcal{A}_\varphi^X$. It is easy to see that $\mathcal{A}_\varphi^X$ must be counter-free, since if it had a counter, the counter must contain a transition with a 1 in the x-component in the original ST-NFA $\mathcal{A}_\psi^{X'}$. But then by our assumption on the structure of $\mathcal{A}_\psi^{X'}$, it would accept non-valid X' models having multiple 1's in the x-component.

From the above lemma it now follows that for a sentence $\varphi \in \mathsf{FO}^{\mathsf{c}}(A)$ we have a counter-free ST-NFA $\mathcal{A}_\varphi$ such that $F(\varphi) = F(\mathcal{A}_\varphi)$. In particular, if we are interested in the alternating function language of φ, we can conjunct φ with the $\mathsf{FO}^{\mathsf{c}}(A)$ formula $\varphi_{alt} = \forall x((\varphi_{disc} \Rightarrow \bigvee_{a \in A_1} Q_a(x)) \wedge (\varphi_{cont} \Rightarrow \bigvee_{a \in A_2} Q_a(x)))$ which forces models to be alternating. The resulting ST-NFA will also be alternating.

Steps (b) to (d) prove that we can go from an arbitrary counter-free ST-NFA $\mathcal{A}$ over the partitioned alphabet (A_1, A_2) to an equivalent $\mathsf{FO}^{\mathsf{c}}(A)$-sentence $\varphi_{\mathcal{A}}$.

Step (b): By Prop. 2, for a counter-free ST-NFA $\mathcal{A}$ over (A_1, A_2) we can give a classical counter-free NFA $\mathcal{B}$ such that $L_{sym}(\mathcal{A}) = L_{sym}(\mathcal{B})$.

Step (c): For a counter-free NFA $\mathcal{B}$, by the McNaughton-Papert result [MP71] we can give an $\mathsf{FO}(A)$ formula ψ, where the logic $\mathsf{FO}(A)$ is the discrete version of $\mathsf{FO}^{\mathsf{c}}(A)$ defined in a similar manner to $\mathsf{LTL}(A)$, such that $L_{sym}(\psi) = L_{sym}(\mathcal{B})$. From Kamp's result for discrete LTL [Kam68], we have an equivalent $\mathsf{LTL}(A)$ formula θ such that $L_{sym}(\psi) = L_{sym}(\theta)$.

Step (d): For a formula θ in $\mathsf{LTL}(A)$ we construct a formula $\mathit{ltl\text{-}ltlc}(\theta)$ in $\mathsf{LTL}^{\mathsf{c}}(A)$ which is such that $F(\mathit{ltl\text{-}ltlc}(\theta)) = \mathit{timing}(L_{sym}(\theta))$.

We will use the abbreviation $\theta_1 U_d\, \theta_2$ to mean that at a point of discontinuity "$\theta_1 U \theta_2$" is true in an untimed sense, and define it to be $(\theta_2 U \theta_2) \vee (\theta_1 U (\theta_{disc} \wedge (\theta_2 \vee (\theta_1 \wedge (\theta_2 U \theta_2)))))$. Symmetrically we use $\theta_1 S_d\, \theta_2$ for $(\theta_2 S \theta_2) \vee (\theta_1 S (\theta_{disc} \wedge (\theta_2 \vee (\theta_1 \wedge (\theta_2 S \theta_2)))))$.

The translation $\mathit{ltl\text{-}ltlc}$ is defined as follows (we use $\widehat{\eta}$ for $\mathit{ltl\text{-}ltlc}(\eta)$ for brevity):

$$
\begin{aligned}
\mathit{ltl\text{-}ltlc}(a) &= a.\\
\mathit{ltl\text{-}ltlc}(\neg\theta_1) &= \neg\widehat{\theta_1}.\\
\mathit{ltl\text{-}ltlc}(\theta_1 \vee \theta_2) &= \widehat{\theta_1} \vee \widehat{\theta_2}.\\
\mathit{ltl\text{-}ltlc}(\theta_1 U \theta_2) &= (\theta_{disc} \Rightarrow (\widehat{\theta_1} U_d\, \widehat{\theta_2})) \wedge\\
&\qquad (\theta_{cont} \Rightarrow (\theta_{cont}\, U (\theta_{disc} \wedge (\widehat{\theta_2} \vee (\widehat{\theta_1} \wedge (\widehat{\theta_1} U_d\, \widehat{\theta_2})))))).\\
\mathit{ltl\text{-}ltlc}(\theta_1 S \theta_2) &= (\theta_{disc} \Rightarrow (\widehat{\theta_1} S_d\, \widehat{\theta_2})) \wedge\\
&\qquad (\theta_{cont} \Rightarrow (\theta_{cont}\, S (\theta_{disc} \wedge (\widehat{\theta_2} \vee (\widehat{\theta_1} \wedge (\widehat{\theta_1} S_d\, \widehat{\theta_2})))))).
\end{aligned}
$$

Lemma 3. *Let θ be an $\mathsf{LTL}(A)$ formula. Let w be a canonical word in A^*. Let $f \in timing(w)$ with a canonical interval representation $(a_0, I_0)\cdots(a_{2n}, I_{2n})$. Then for all $i \in \{0, \cdots, 2n\}$ and for all $t \in I_i$, we have $w, i \models \theta \iff f, t \models ltl\text{-}ltlc(\theta)$.* □

From the above lemma it follows that $F(ltl\text{-}ltlc(\theta)) = timing(L_{sym}(\theta))$.

Step (e): Using Kamp's theorem [Kam68] for a given $\mathsf{LTL}^{\mathsf{c}}(A)$ formula $\widehat{\theta}$ we can give an equivalent $\mathsf{FO}^{\mathsf{c}}(A)$ formula φ such that $F(\widehat{\theta}) = F(\varphi)$.

To summarize this direction of the proof: given a counter-free ST-NFA $\mathcal{A}$ over (A_1, A_2) by steps (b) and (c) we have an $\mathsf{LTL}(A)$ formula θ such that $L_{sym}(\mathcal{A}) = L_{sym}(\theta)$. By steps (c) and (d) we have an $\mathsf{FO}^{\mathsf{c}}(A)$ formula $\varphi_{\mathcal{A}}$ such that $timing(L_{sym}(\theta)) = F(\varphi_{\mathcal{A}})$. It follows that $F(\mathcal{A}) = F(\varphi_{\mathcal{A}})$. Further, by the alternating nature of the symbolic language of $\mathcal{A}$ it follows that the function models and alternating function models of $\varphi_{\mathcal{A}}$ are the same.

This completes the proof of Theorem 1. □

5 Counter-Free *CIDA*'s and $\mathsf{TFO}^{\mathsf{c}}$

We can now prove the main result of this paper which is a general characterization of timed first-order definable languages (again, for a natural first-order logic based on input-determined operators) via counter-free *CIDA*'s.

We recall the definition of the *continuous timed first-order logic* ($\mathsf{TFO}^{\mathsf{c}}$) based on (Σ, Op) from [CDP06]. The syntax of the logic $\mathsf{TFO}^{\mathsf{c}}(\Sigma, Op)$ is given by:

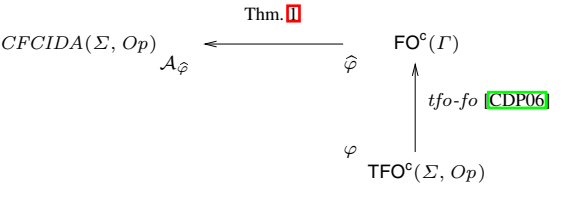
$$\varphi ::= Q_a(x) \mid \Delta^I(x) \mid x < y \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \exists x\varphi,$$

where $a \in \Sigma$, $\Delta \in Op$, $I \in \mathcal{I}_{\mathbb{Q}_{\geq 0}}$, and x and y are first-order variables.

The logic is interpreted over timed words in $T\Sigma^*$, in a way similar to the logic FO^{c}. Given a formula $\varphi \in \mathsf{TFO}^{\mathsf{c}}(\Sigma, Op)$, a timed word $\sigma = (a_1, t_1)\cdots(a_n, t_n)$ in $T\Sigma^*$, and an interpretation $\mathbb{I}$ with respect to σ, which maps a first order variable x to $t \in [0, dur(\sigma)]$, we say $\sigma, \mathbb{I} \models \mathrm{Q}_a(x)$ iff $\exists i : a_i = a$, and $t_i = \mathbb{I}(x)$; $\sigma, \mathbb{I} \models \Delta^I(x)$ iff $\Delta(\sigma, \mathbb{I}(x)) \cap I \neq \emptyset$; and the rest of the cases are similar to that of the logic FO^{c} defined in the previous section. For a sentence φ in $\mathsf{TFO}^{\mathsf{c}}(\Sigma, Op)$, the timed language defined by φ, denoted $L(\varphi)$, is defined to be $\{\sigma \in T\Sigma^* \mid \sigma \models \varphi\}$.

Theorem 2. *Let Σ be an alphabet and Op a set of finitely varying input-determined operators over Σ. A timed language $L \subseteq T\Sigma^*$ is definable by a $\mathsf{TFO}^{\mathsf{c}}(\Sigma, Op)$ sentence iff it is definable by a CFCIDA over (Σ, Op).*

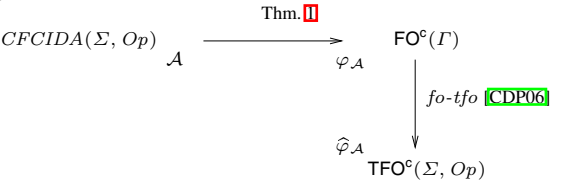
Proof. We first show how to go from $\mathsf{TFO}^{\mathsf{c}}$ to *CFCIDA*. The route we take is shown in the figure alongside:



Let φ be a $\mathsf{TFO}^{\mathsf{c}}(\Sigma, Op)$ sentence. Then there is a proper symbolic alphabet (Γ_1, Γ_2) over (Σ, Op) and a $\mathsf{FO}^{\mathsf{c}}(\Gamma_1 \cup \Gamma_2)$ sentence $\widehat{\varphi}$ such that $L(\varphi) = tw(F(\widehat{\varphi}))$. The symbolic alphabet (Γ_1, Γ_2) is based on the set of guards $\{\Delta^I | \Delta^I(x)$ is a subformula of $\varphi\}$. The formula $\widehat{\varphi}$ is then obtained from φ by replacing each $Q_a(x)$ by $\bigvee_{(a,h)\in\Gamma} Q_{(a,h)}(x)$ and $\Delta^I(x)$ by $\bigvee_{(c,h)\in\Gamma, \Delta^I \in h} Q_{(c,h)}(x) \vee \bigvee_{h\in\Gamma, \Delta^I\in h} Q_h(x)$, where $\Gamma = \Gamma_1 \cup \Gamma_2$, and taking its conjunction with φ_{fcp}, which is satisfied by functions whose untimings are fully canonical proper words.

From Theorem 1, we have a counter-free ST-NFA $\mathcal{A}_{\widehat{\varphi}}$ over (Γ_1, Γ_2) such that $F(\mathcal{A}_{\widehat{\varphi}}) = F(\widehat{\varphi})$. By construction of $\widehat{\varphi}$ it follows that $\mathcal{A}_{\widehat{\varphi}}$ is a fully canonical proper $CIDA$ over (Σ, Op), and hence a $CFCIDA$ over (Σ, Op). Since $L(\mathcal{A}_{\widehat{\varphi}}) = tw(F(\mathcal{A}_{\widehat{\varphi}}))$ $= tw(F(\widehat{\varphi})) = L(\varphi)$, we are done.

In the converse direction, the route we follow is:



Let $\mathcal{A}$ be a $CFCIDA$ over (Σ, Op). Thus $\mathcal{A}$ is a counter-free ST-NFA over a proper alphabet (Γ_1, Γ_2) based on (Σ, Op), which accepts a fully canonical function language. By Theorem 1 we have a $\mathsf{FO}^{\mathsf{c}}(\Gamma)$ sentence $\varphi_{\mathcal{A}}$ (where $\Gamma = \Gamma_1 \cup \Gamma_2$), such that $F(\varphi_{\mathcal{A}}) = F(\mathcal{A})$. We now use the translation *fo-tfo* from [CDP06] which simply "unpacks" a formula φ in $\mathsf{FO}^{\mathsf{c}}(\Gamma)$ to a formula $\widehat{\varphi}$ in $\mathsf{TFO}^{\mathsf{c}}(\Sigma, Op)$ such that $L(\widehat{\varphi}) = tw(F(\varphi))$. Thus we take $\widehat{\varphi}_{\mathcal{A}}$ to be $fo\text{-}tfo(\varphi_{\mathcal{A}})$, and we have that $L(\mathcal{A}) = tw(F(\mathcal{A})) = tw(F(\varphi_{\mathcal{A}})) = L(\widehat{\varphi}_{\mathcal{A}})$. □

6 Counter-Free Recursive *CIDA*'s

Our aim is now to extend the counter-free characterization of first-order definable timed languages to "recursive" (or "hierarchical") first-order logic and $CIDA$'s. This will give us a counter-free $CIDA$ characterization for many of the timed temporal logics defined in the literature, including $\mathsf{MTL}^{\mathsf{c}}$+Past and MITL.

We begin with a few preliminaries, mostly from [CDP06]. A *floating timed word* over Σ is a pair (σ, t), where σ in $T\Sigma^*$ and $t \in [0, dur(\sigma)]$. We denote the set of floating timed words over Σ by $FT\Sigma^*$. We will represent a floating word over Σ as timed word over the alphabet $\Sigma' = (\Sigma \cup \{\epsilon\}) \times \{0, 1\}$. For a timed word σ' over Σ', let σ denote the timed word obtained from σ' by projecting away the $\{0, 1\}$ component from each pair and then dropping any ϵ's in the resulting word. Then a timed word σ' over Σ' which contains exactly one symbol from $(\Sigma \cup \{\epsilon\}) \times \{1\}$, and whose last symbol is from $\Sigma \times \{0, 1\}$, represents the floating timed word (σ, t), where t is the time of the unique action which has a 1-extension. We use fw to denote the (partial) map which given a timed word σ' over Σ' returns the floating word (σ, t) represented by it, and extend it to apply to timed languages over Σ' in the natural way.

Let Σ be an alphabet and Op be a set of input determined operators. Given $\Delta \in Op$, we use the notation Δ' for the operator over Σ' with the semantics $\Delta'(\sigma', t) = \Delta(\sigma, t)$. We use the notation Op' to denote the set $\{\Delta' \mid \Delta \in Op\}$. We now define a *floating*

CIDA over (Σ, Op) to be a *CIDA* over (Σ', Op'). We define the floating language of a floating *CIDA* $\mathcal{B}$, denoted $L^{fl}(\mathcal{B})$, as $fw(L(\mathcal{B}))$.

A *recursive* input-determined operator Δ over an alphabet Σ is a partial function from $(2^{FT\Sigma^*} \times T\Sigma^* \times \mathbb{R}_{\geq 0})$ to $2^{\mathbb{R}_{\geq 0}}$, which is defined for tuples (M, σ, t) where M is a floating language over Σ, $\sigma \in T\Sigma^*$, and $t \in [0, dur(\sigma)]$. Thus, given a floating language M, we obtain an input-determined operator Δ_M whose semantics is given by $\Delta_M(\sigma, t) = \Delta(M, \sigma, t)$. For a floating *CIDA* $\mathcal{B}$, we write $\Delta_\mathcal{B}$ for the operator $\Delta_{L^{fl}(\mathcal{B})}$.

We call a floating language M over Σ finitely varying, if for each timed word σ, the characteristic function of the set $pos(M, \sigma) = \{t \mid (\sigma, t) \in M\}$ is finitely varying in $[0, dur(\sigma)]$. We say a recursive operator Δ is *finitely varying* if for every finitely varying floating language M, the operator Δ_M is finitely varying.

We are now ready to define the recursive version of our *CIDA*'s. We define the class of *recursive CIDA*'s (*rec-CIDA*'s), and the class of *recursive floating CIDA*'s (*frec-CIDA*'s) over an alphabet Σ and a set of recursive operators Rop based on Σ, as the union over $i \in \mathbb{N}$, of level-i *rec-CIDA*'s over (Σ, Rop) and level i *frec-CIDA*'s over (Σ, Rop), which are defined inductively below:

- A level-0 *rec-CIDA* over (Σ, Rop) is a *CIDA* $\mathcal{A}$ over Σ that uses only the guard $\top$. It accepts the timed language accepted by $\mathcal{A}$ viewed as a *CIDA* – i.e. $L(\mathcal{A})$. A level-0 *frec-CIDA* over (Σ, Rop) is a floating *CIDA* $\mathcal{B}$ over Σ which uses only the guard $\top$. It accepts the floating language $L^{fl}(\mathcal{B})$ (i.e by viewing it as a floating *CIDA* over Σ).
- A level-$i+1$ *rec-CIDA* over (Σ, Rop) is a *CIDA* $\mathcal{A}$ over Σ and finite set of operators Op of the form $\Delta_\mathcal{B}$, where $\Delta \in Rop$ and $\mathcal{B}$ is a level-i or less *frec-CIDA* over (Σ, Rop). We require that $\mathcal{A}$ uses at least one operator of the form $\Delta_\mathcal{B}$ wit $\mathcal{B}$ a level-i *frec-CIDA*. The timed language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is defined to be the timed language accepted by $\mathcal{A}$ viewed as a *CIDA* over (Σ, Op).
  A level-$i+1$ *frec-CIDA* over (Σ, Rop) is a floating *CIDA* $\mathcal{B}$ over Σ and finite set of operators Op of the form $\Delta_\mathcal{C}$, where $\Delta \in Rop$ and $\mathcal{C}$ is a level-i or less *frec-CIDA* over (Σ, Rop). We require that $\mathcal{B}$ uses at least one operator of the form $\Delta_\mathcal{C}$ wit $\mathcal{C}$ a level-i *frec-CIDA*. The floating language $L^{fl}(\mathcal{B})$ accepted by $\mathcal{B}$ is defined to be the floating language accepted by $\mathcal{B}$ viewed as a floating *CIDA* over (Σ, Op).

We now define the counter-free versions of these automata, by induction on the level in which they occur. A level-0 *rec-CIDA* (respectively *frec-CIDA*) is counter-free if the underlying ST-NFA is counter-free. A level-$i+1$ *rec-CIDA* (resp. *frec-CIDA*) is counter-free if it only uses operators of the form $\Delta_\mathcal{B}$ where $\mathcal{B}$ is a counter-free *frec-CIDA* of level-i or less, and the underlying ST-NFA is counter-free.

We extend these definitions to proper *rec-CIDA*'s and *frec-CIDA*'s in the obvious way. We define the class of counter-free *rec-CIDA* languages over (Σ, Rop), denoted *rec-CFIDA*(Σ, Rop), to be the class of timed languages definable by counter-free fully canonical proper *rec-CIDA*'s over (Σ, Rop).

We now introduce the recursive version of TFO^c. Given an alphabet Σ and a set of recursive operators Rop, the set of formulas of rec-TFO$^c(\Sigma, Rop)$ are defined inductively as: $\varphi ::= Q_a(x) \mid \Delta^I_\psi(x) \mid x < y \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\varphi$, where $a \in \Sigma$, $\Delta \in Rop$, $I \in \mathcal{I}_{\mathbb{Q}_{\geq 0}}$ and ψ is a rec-TFO^c formula with a single free variable z. The

rec-TFO^c formulas are interpreted similar to TFO^c formulas where the operator Δ_ψ is defined by $\Delta_\psi(\sigma,t) = \Delta(L^{ft}(\psi),\sigma,t)$ and $L^{ft}(\psi) = \{(\sigma,t) \mid \sigma,[t/z] \models \psi\}$. A rec-TMSO$^c(\Sigma, Rop)$ sentence φ defines the language $L(\varphi) = \{\sigma \in T\Sigma^* \mid \sigma \models \varphi\}$.

Using a similar technique to the proof of Theorem 2 we can show that rec-TFO^c-definable and *rec-CFIDA*-definable languages are the same:

Theorem 3. *Let Σ be an alphabet and Rop be a set of finitely-varying recursive operators based on Σ. Then a timed language $L \subseteq T\Sigma^*$ is definable by a* rec-TFO$^c(\Sigma, Rop)$ *sentence iff it is definable by a rec-CFIDA over (Σ, Rop).* □

We recall the definition of the recursive timed temporal logic based on (Σ, Rop) from [CDP06], denoted rec-TLTL$^c(\Sigma, Rop)$. The syntax of the logic is given by

$$\theta ::= a \mid \Delta_\theta^I \mid (\theta U \theta) \mid (\theta S \theta) \mid \neg\theta \mid (\theta \vee \theta),$$

where $a \in \Sigma$, $\Delta \in Rop$ and $I \in \mathcal{I}_{\mathbb{Q}_{\geq 0}}$. The logic is interpreted over timed words in a manner similar to TLTL^c+Past, where the operator Δ_θ is defined by $\Delta_\theta(\sigma,t) = \Delta(L^{ft}(\theta),\sigma,t)$, and $L^{ft}(\theta) = \{(\sigma,t) \mid \sigma,t \models \theta\}$. From [CDP06] we know that:

Theorem 4 ([CDP06]). *Let Σ be an alphabet and Rop be a set of finitely-varying recursive operators based on Σ. Then a timed language $L \subseteq T\Sigma^*$ is definable by a* rec-TFO$^c(\Sigma, Rop)$ *sentence iff it is definable by a* rec-TLTL$^c(\Sigma, Rop)$ *formula.* □

Putting Theorems 3 and 4 together we obtain counter-free *CIDA* characterizations for many timed temporal logics based on input-determined operators, proposed in the literature. In particular we obtain a counter-free *CIDA* characterization for the logic MTL^c+Past (with past operators). Recall that the syntax of the logic MTL^c+Past(Σ) is:

$$\theta ::= a \mid (\theta U_I \theta) \mid (\theta S_I \theta) \mid \neg\theta \mid (\theta \vee \theta),$$

where $a \in \Sigma$ and $I \in \mathcal{I}_{\mathbb{Q}_{\geq 0}}$. The logic is interpreted over timed words in $T\Sigma^*$, and the modalities U_I (and symmetrically S_I) is as follows:

$$\sigma,t \models \theta U_I \eta \text{ iff } \exists t' \geq t : t' - t \in I, \sigma,t' \models \eta, \text{ and } \forall t'' : t < t'' < t', \sigma,t'' \models \theta.$$

We recall that MTL^c+Past was shown to be expressively equivalent to the logic rec-TLTL$^c(\Sigma, \{\diamond, \diamondminus\})$ in [CDP06], where the recursive operators $\diamond$ and $\diamondminus$ are defined as $\diamond(M,\sigma,t) = \{t' - t \mid t' \geq t, t \in pos(M,\sigma)\}$ and $\diamondminus(M,\sigma,t) = \{t - t' \mid t' \leq t, t \in pos(M,\sigma)\}$. Thus we have:

Theorem 5. *The class of timed languages definable by rec-CFIDA$(\Sigma, \{\diamond, \diamondminus\})$ and* MTL^c+Past(Σ) *are the same.*

Restricting to non-singular intervals we obtain a similar result for the logic MITL^c+Past [AFH96].

7 Ultimate Stability of MTL^c+Past

In section 6 we showed that every MTL^c+Past language is recognized by a recursive counter-free *CIDA*. We will now use this characterization to show the *ultimate stability*

of MTL^c+Past with respect to periodic sequences of timed words. In this section we assume that Σ is an alphabet and *Rop* a set of finitely-varying recursive operators.

A *periodic sequence* of timed words $\langle\sigma_i\rangle$ is of the form $uw, uvw, uv^2w, \cdots$ for some timed words u, v and w in $T\Sigma^*$. We represent $\langle\sigma_i\rangle$ above via the triple (u, v, w). A language of timed words $L \subseteq T\Sigma^*$ is said to be *ultimately stable* w.r.t. a periodic sequence $\langle\sigma_i\rangle$ if there exists $i_0 \in \mathbb{N}$ such that either $\forall i \geq i_0,\ \sigma_i \in L$ or $\forall i \geq i_0,\ \sigma_i \notin L$. The language L is said to be *ultimately stable* if it is ultimately stable w.r.t. all periodic sequences of timed words.

Theorem 6. *Let φ be an* MTL^c+Past *formula. Then $L(\varphi)$ is ultimately stable.*

The rest of this section is devoted to the proof of the above theorem. By theorem 5 we just need to show that all languages recognized by *rec-CFIDA* are ultimately stable.

We first introduce the concept of middle zone: it represents the set of time points "in the middle" of a timed word. A *middle zone* is a couple $Z = (l, r)$ with $l, r \in \mathbb{R}$. Given a timed word w we define $Z(w) = (l,\ dur(w) - r)$.

Fig. 1. A middle zone

A floating language $L \subseteq T\Sigma^* \times \mathbb{R}$ is said to be *well-behaved* w.r.t. a periodic sequence $\langle\sigma_i\rangle = (u, v, w)$ if there exist a middle zone Z and an index i_0 such that the following conditions hold:

$$\begin{aligned} &\forall i\ \forall i' \geq i\ \forall t \in Z(\sigma_i),\ (\sigma_i, t) \in L \Leftrightarrow (\sigma_i, t + dur(v)) \in L \text{ and} \\ &\quad (\sigma_i, t) \in L \Leftrightarrow (\sigma_{i'}, t) \in L. \end{aligned} \tag{1}$$

$$\forall i \geq i_0\ \forall i' \geq i\ \forall t < l,\ (\sigma_i, t) \in L \Leftrightarrow (\sigma_{i'}, t) \in L. \tag{2}$$

$$\forall i \geq i_0\ \forall i' \geq i\ \forall t < r,\ (\sigma_i, dur(\sigma_i) - t) \in L \Leftrightarrow (\sigma_{i'}, dur(\sigma_{i'}) - t) \in L. \tag{3}$$

A floating language L is said to be *well-behaved* if it is *well-behaved* w.r.t. all periodic sequences. Note that a guard Δ^I defines a floating language given by $\{(\sigma, t) \mid \sigma, t \models \Delta^I\}$. We say that a floating *CFCIDA* (resp. a guard) is *well-behaved* w.r.t. a periodic sequence if its associated language is. Similarly we say that a floating *CFCIDA* (resp. a guard) is *well-behaved* if its associated language is.

Given a proper symbolic alphabet Γ and a timed word σ we denote by γ_σ^Γ the unique symbolic word $\gamma \in \Gamma^*$ such that $\sigma \in tw(\gamma)$. The proofs of the two lemmas below can be found in [CDP07].

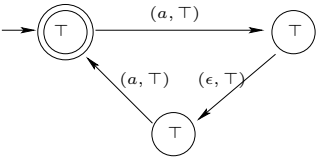
Lemma 4. *Let G be a finite set of guards and Γ be a proper symbolic alphabet over (Σ, Rop) based on G. Let $\langle\sigma_i\rangle$ be a periodic sequence. If for all $g \in G$, g is well-behaved w.r.t. $\langle\sigma_i\rangle$ then there exists an integer i_0 and $\gamma_1, \gamma_2, \gamma_3 \in \Gamma^*$ such that for all $i \geq i_0$, $\gamma_{\sigma_i}^\Gamma = \gamma_1\gamma_2^{i-i_0}\gamma_3$.* □

Note that this lemma shows that if for all $g \in G$, g is well-behaved w.r.t. $\langle \sigma_i \rangle$ then any floating automaton based on G is well-behaved w.r.t. $\langle \sigma_i \rangle$.

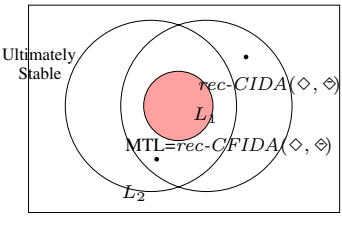
Lemma 5. *Let $\mathcal{B}$ be a proper canonical fCFIDA over (Σ, Rop) and I an interval. If $\mathcal{B}$ is well-behaved, then the guards $\Diamond_B^I$ and $\Diamond\!\!\!\!-_B^I$ are also well-behaved.* □

Returning to the proof of theorem 6: let L be recognized by some *rec-CFIDA* $\mathcal{A}$ and $\langle \sigma_i \rangle$ be a periodic sequence. Let Γ be the proper symbolic alphabet of $\mathcal{A}$; by lemma 4 and 5 there exists an integer i_0 and $\gamma_1, \gamma_2, \gamma_3 \in \Gamma^*$ such that for all $i \geq i_0$, $\gamma^{\Gamma}_{\sigma_i} = \gamma_1 \gamma_2^{i-i_0} \gamma_3$. Let n be the number of states of $\mathcal{A}$. γ_2 cannot be a counter for $\mathcal{A}$ so for i greater than $i_0 + n$, the run of $\mathcal{A}$ on σ_i ends in the same state. Thus L is ultimately stable w.r.t. $\langle \sigma_i \rangle$. □

We justify here why *CFCIDA*'s were defined to include only fully canonical proper words. Had we allowed words which are not fully canonical, *CFCIDA*'s would not have been equivalent to TFO^c. Alongside is a proper *CIDA* which is not fully canonical but the underlying ST-NFA is counter-free. It accepts the timed language L_1 consisting of timed words with even number of a's. This language is not ultimately stable with respect to the periodic sequence $(\epsilon, (a, 1), \epsilon)$, and hence is not definable in MTL^c+Past and therefore not definable in $\mathsf{TFO}^c(\{a\}, \{\Diamond_a\})$.



We also note that, unlike classical LTL, ultimate stability of a *rec-CIDA*$(\Diamond, \Diamond\!\!\!\!-)$ language is not a sufficient condition for MTL^c+Past recognizability. Consider the timed language L_2 consisting of timed words ending with an a at time 1 and having even number of b's in the interval $(0, 1)$. This language is recognized by a *rec-CIDA* over $\{\Diamond\}$. However it can be shown to be inexpressible in MTL^c+Past and hence not recognized by a *rec-CFIDA* over $\{\Diamond\}$. Nevertheless it is trivially ultimately stable.



The Venn diagram alongside shows the different classes of timed languages.

References

[AFH94] Alur, R., Fix, L., Henzinger, T.A.: A Determinizable Class of Timed Automata. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 1–13. Springer, Heidelberg (1994)

[AFH96] Alur, R., Feder, T., Henzinger, T.A.: The Benefits of Relaxing Punctuality. Journal of the ACM 43(1), 116–146 (1996)

[CDP06] Chevalier, F., D'Souza, D., Prabakhar, P.: On continuous timed automata with input-determined guards. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 369–380. Springer, Heidelberg (2006)

[CDP07] Chevalier, F., D'Souza, D., Prabhakar, P.: Counter-free input-determined timed automata. Technical Report IISc-CSA-TR-, -1, Indian Institute of Science, Bangalore 560012, India (January 2007), URL http://archive.csa.iisc.ernet.in/TR/2007/1/

[DM05] D'Souza, D., Mohan, M.R.: Eventual Timed Automata. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 322–334. Springer, Heidelberg (2005)

[DT04] 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)

[HRS98] Henzinger, T.A., Raskin, J.-F., Schobbens, P.-Y.: The Regular Real-Time Languages. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 580–591. Springer, Heidelberg (1998)

[Kam68] Kamp, J.A.W.: Tense Logic and the Theory of Linear Order. PhD thesis, University of California, Los Angeles, California (1968)

[Koy90] Koymans, R.: Specifying Real-Time Properties with Metric Temporal Logic. Real-Time Systems 2(4), 255–299 (1990)

[MP71] McNaughton, R., Papert, S.: Counter-Free Automata. MIT Press, Cambridge, MA (1971)

[OW05] Ouaknine, J., Worrell, J.: On the Decidability of Metric Temporal Logic. In: LICS, pp. 188–197. IEEE Computer Society, Los Alamitos (2005)

[PD06] 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)

[RS99] Raskin, J.-F., Schobbens, P.-Y.: The logic of event clocks: decidability, complexity and expressiveness. Automatica 4(3), 247–282 (1999)

Towards Budgeting in Real-Time Calculus: Deferrable Servers

Pieter J.L. Cuijpers and Reinder J. Bril

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science
Den Dolech 2, 5600 MB Eindhoven, The Netherlands

Abstract. Budgeting of resources is an often used solution for guaranteeing performance of lower priority tasks. In this paper, we take a formal approach to the modeling of a deferrable server budgeting strategy, using real-time calculus. We prove a scheduling theorem for deferrable servers, and as a corollary show that an earlier claim of Davis and Burns, that periodic servers dominate deferrable servers with respect to schedulability, no longer holds when the context of the comparison is slightly generalized.

1 Introduction

One of the main scheduling approaches in real-time computing systems is given by fixed-priority preemptive scheduling (FPPS) [5,6]. The approach is founded on a fixed-priority scheduling theory, supported by a suite of open software standards, commercially available schedulability analysis tools and real-time operating systems, and adopted by leading companies and institutions world-wide.

Scheduling a set of real-time tasks sharing a resource gives rise to the so-called *temporal interference* problem, i.e. a malfunctioning task may cause other tasks to fail to meet their time constraints. An often used solution for this problem is to introduce *resource budgets* for tasks, which provide temporal protection between tasks by guaranteeing a minimal amount of resources [8]. Those budgets are often implemented using so-called *servers* that dispatch the available resources to the tasks that are appointed to them.

In general, a server for a shared processing resource, such as a CPU, is characterized by a *capacity* and a *replenishment period* [1]. The capacity is the maximum amount of resources (i.e. the maximum amount of processing time) that a server can provide to its associated tasks during its replenishment period. The replenishment period is the minimum time between replenishments of the capacity. Servers typically differ with respect to the amount and moment in time of the replenishments and to the preservation of the remaining capacity when none of the appointed tasks is ready to use it.

In this paper, we consider so-called *deferrable servers* [10], which have been studied as implementations of resource budgets in [2,9], amongst others. We focus on [2], because its results improve on earlier work. Deferrable servers are

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 98–113, 2007.

replenished *periodically*, at fixed intervals of time, and have the following preservation policy. When a deferrable server has access to the shared resource, its capacity is provided if one of the tasks is ready to use it. If the tasks are not ready to use it, the deferrable server suspends its access to the resource, preserving its remaining capacity. Capacity can be preserved until the end of the replenishment period. At the end of the server's period any remaining capacity is lost.

Our main contribution, is a formal model of deferrable servers using the real-time calculus of [11]. Real-time calculus is a branch of network calculus [4], which uses max-plus algebra for the algebraic analysis of real-time systems. Our specific interest in formalizing the behavior of deferrable servers using this calculus, was due to a suspicion that the worst-case response time analysis of deferrable servers in [2] becomes pessimistic, rather than exact, in a slightly generalized context. Using our real-time calculus model, we derive a schedulability theorem for deferrable servers with a single task. Application of this theorem for a deferrable server with highest priority leads to an example showing that, in the absence of a low-priority soft real-time task, the analysis in [2] indeed becomes pessimistic, rather than exact. As a consequence, the applicability of deferrable servers may be better than was suggested in [2].

This paper is structured as follows. First, in Section 2, we recapitulate the real-time calculus theory of [11]. Next, in Section 3 we explain how to model servers in this theory, and extend the theory with a model of a deferrable server. Our schedulability theorem is the topic of Section 4. The consequences of our schedulability theorem are discussed in Section 5, where we compare it with the results of [2]. Section 6 summarizes the main contributions of the paper and suggests directions for future research.

2 Real-Time Calculus Preliminaries

In this section, we recall part of the real-time calculus theory of [11]. The starting point of this calculus, is to consider cumulative requests streams $R(t) : \mathbb{R} \to \mathbb{R}$ (from time to amount of requested resources) and cumulative resource streams $C(t) : \mathbb{R} \to \mathbb{R}$ (from time to amount of available resources) in a system. The request stream models the total amount of requested tasks that have entered the system at a certain time, while the resource stream models the total amount of processing power that has been offered to a server. The total amount of tasks that have been processed is modeled by a cumulative request stream $R'(t)$, while the total amount of resources that remains unused is processed by a cumulative resource stream $C'(t)$. For convenience we choose $C(t) = R(t) = C'(t) = R'(t) = 0$ whenever $t < 0$, reflecting that the system is turned on at $t = 0$.

Processing of a sequence of tasks can be depicted as in figure 1. It is obvious, that the total amount of tasks processed at time t can never exceed the amount of tasks processed already at a time $u \leq t$, plus the amount of resources offered between u and t. Furthermore, the amount of processed tasks can never be

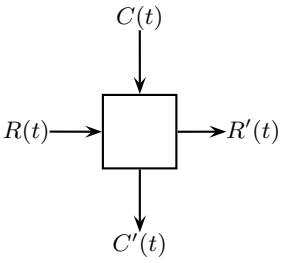


Fig. 1. Basic processing in Real-time Calculus

more than the amount of requested tasks. This lower bound on the output R' is captured in the following formula[1].

$$R'(t) \leq \big(R'(u) + C(t) - C(u)\big) \wedge R(t).$$

With a little calculation we can rewrite this into:

$$R'(t) \leq \inf_{u \leq t} \{R(u) + C(t) - C(u)\}.$$

Now, if we furthermore assume that *tasks may be buffered until resources become available for it*, and that a server is *eager* in the sense that it processes each task as soon as resources are available for it, we can derive a lower bound on $R'(t)$ as well. We define t_0 as the latest point before t at which the resource buffer was empty, assuming $R(0) = R'(0)$ for convenience to show there exists such a point.

$$t_0 \triangleq \sup\{\tau \leq t \mid R(\tau) = R'(\tau)\}$$

Between t_0 and t, the task buffer is always non-empty, which means that all resources that arrive are used for processing tasks. So we find the equality:

$$R'(t) = \big(R'(t_0) + C(t) - C(t_0)\big) \wedge R(t).$$

For a right-continuous resource stream $R(t)$ we then find $R(t_0) = R'(t_0)$ (using the definition of t_0) and thus:

$$\begin{aligned} R'(t) &= (R(t_0) + C(t) - C(t_0)) \wedge R(t), \\ &\geq \inf_{u \leq t} \{R(u) + C(t) - C(u)\} \wedge R(t), \\ &= \inf_{u \leq t} \{R(u) + C(t) - C(u)\}. \end{aligned}$$

In conclusion, $R'(t)$ is exactly determined by

[1] In this paper, we use $x \wedge y$ to denote the minimum of x and y, and $\inf_u\{f(u)\}$ to denote the infimum of $f(u)$ over u. Furthermore, $x \vee y$ denotes the maximum of x and y, and $\sup_u\{f(u)\}$ denotes the supremum of $f(u)$ over u.

$$R'(t) = \inf_{u \le t} \{R(u) + C(t) - C(u)\}.$$

According to [4] the same formula holds if $R(t)$ is left-continuous, but the proof is more complicated. The amount of resources that is left unused can be found easily by subtracting what is used from what is delivered.

$$\begin{aligned} C'(t) &= C(t) - R'(t), \\ &= C(t) - \inf_{u \le t} \{R(u) + C(t) - C(u)\}, \\ &= \sup_{u \le t} \{C(u) - R(u)\}. \end{aligned}$$

In figure 2 we have depicted an example of a task $R(t) = 3 \cdot \lceil \frac{t}{3} \rceil \vee 0$, modeling the arrival of three requests every three time-units. This task is serviced by a resource $C(t) = 2 \cdot t \vee 0$, which brings continuous service to a task at two resource-units per time-unit. Note, that by definition $C'(t)$ is flat whenever $R'(t)$ rises, since eager processing requires that all incoming resources are used as long as the input buffer is non-empty. Vice versa, $R'(t)$ is flat when $C'(t)$ rises, for the same reason.

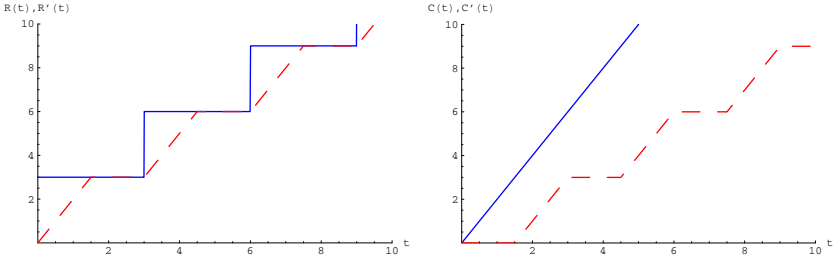


Fig. 2. Requests and resources in RTC (inputs straight, outputs dashed)

3 Deferrable Server Model

In the previous section, we have shown how the processing of a single sequence of tasks can be modeled using real-time calculus. But from a more high-level point of view, the same equations can also be used to represent a server that dispatches resources to a number of tasks. The input stream R_S of the server then is the sum of the input streams of the appointed tasks (whenever one of its tasks has unfinished requests, the server must dispatch incoming resources to it) and the output stream R'_S does not represent finished tasks, but represents resources that have been reserved for the tasks operating under the server. This scheme is depicted in figure 3, where the block labeled S represents a server and

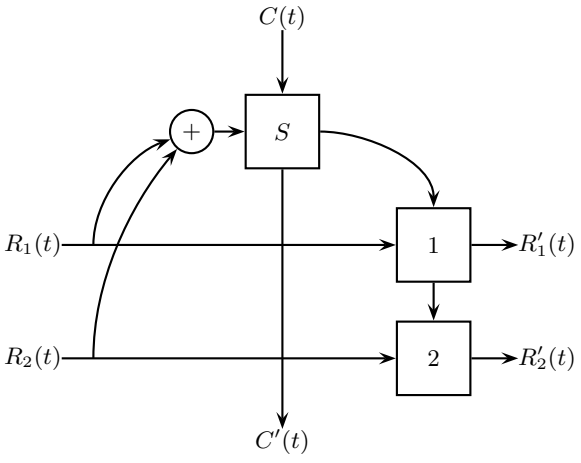


Fig. 3. Server modeling in real-time calculus

blocks 1 and 2 represent tasks that are dispatched to this server. So, we have $R_S = R_1 + R_2$, $C_1 = R'_S$ and $C_2 = C'_1$.

The insight that the equations for a server and for a task are the same, albeit with a different interpretation of the meaning of the variables, still holds when we shift our focus to servers with a budgeting strategy. In other words, this hierarchical way of modeling allows us to model the budgeting of a single task, and use this model for a budgeting server as well. For a budgeting server, we still assume that tasks are processed eagerly, and that tasks are buffered while waiting for resources to arrive. Therefore, the previously derived upper bound on $R'(t)$ is still valid.

$$R'(t) \leq \inf_{u \leq t} \{R(u) + C(t) - C(u)\}.$$

But, additionally, a deferrable server periodically limits the resources it provides to a maximum capacity Q. Replenishment of the capacity takes place at the start of each period T, which at a time t is determined (right-continuously) by $T \cdot \lceil \frac{t}{T} - 1 \rceil$. The output $R'(t)$ of a deferrable server cannot be greater than the output $R'\left(T \cdot \lceil \frac{t}{T} - 1 \rceil\right)$ at the start of the period plus the processing capacity Q. So, we refine the upper bound to capture this behavior.

$$\begin{aligned} R'(t) \leq & \inf_{u \leq t} \{R(u) + C(t) - C(u)\} \\ & \wedge R'\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right) + Q \\ & \wedge R'\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right) + C(t) - C\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right) \end{aligned}$$

Furthermore, for $t_0 \triangleq \sup\{\tau \leq t \mid R(\tau) = R'(\tau)\} \geq 0$ we find

$$\begin{aligned} R'(t) &= R(t_0) + C(t) - C(t_0) \\ &\quad \wedge\ R'\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right) + Q \\ &\quad \wedge\ R'\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right) + C(t) - C\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right) \\ &\geq \inf_{u \leq t} \{R(u) + C(t) - C(u)\} \\ &\quad \wedge\ R'\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right) + Q \\ &\quad \wedge\ R'\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right) + C(t) - C\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right). \end{aligned}$$

What results is a recursive specification for the output of the deferrable server

$$\begin{aligned} R'(t) &= \inf_{u \leq t} \{R(u) + C(t) - C(u)\} \\ &\quad \wedge\ R'\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right) + Q \\ &\quad \wedge\ R'\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right) + C(t) - C\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right). \end{aligned}$$

To prove that this recursive specification has a unique solution for $R'(t)$, we rewrite it to

$$\begin{aligned} R'(t+T) &= \inf_{u \leq t+T} \{R(u) + C(t+T) - C(u)\} \\ &\quad \wedge\ R'\left(T \cdot \left\lceil \frac{t}{T} \right\rceil\right) + Q \\ &\quad \wedge\ R'\left(T \cdot \left\lceil \frac{t}{T} \right\rceil\right) + C(t+T) - C\left(T \cdot \left\lceil \frac{t}{T} \right\rceil\right). \end{aligned}$$

From this representation it is clear that, if the solution of the recursive specification is unique upto a point t, then it is unique upto $t+T$. Furthermore, we find uniqueness for $t < 0$ where we have $R'(t) = 0$. So, the solution is unique upto 0, and with induction upto $n \cdot T$ for any $n \in \mathbb{N}$. We may conclude that $R'(t)$ is well defined, and may even solve the recursive specification to find:

$$\begin{aligned} R'(t) = \inf_{n \in \mathbb{N}} \inf_{u \leq t \wedge T \cdot \left\lceil \frac{t}{T} - n \right\rceil} \Bigg\{ & R(u) + C(t \wedge T \cdot \left\lceil \frac{t}{T} - n \right\rceil) - C(u) \\ & + \sum_{m=0}^{n-1} Q \wedge \left(C(t \wedge T \cdot \left\lceil \frac{t}{T} - m \right\rceil) - C(T \cdot \left\lceil \frac{t}{T} - m - 1 \right\rceil) \right) \Bigg\}. \end{aligned}$$

As before, we find $C'(t)$ by subtracting what is used from what is delivered. With a little calculation, we obtain:

$$C'(t) = \sup_{n\in\mathbb{N}} \sup_{u\le t\wedge T\cdot\left\lceil \frac{t}{T}-n\right\rceil} \left\{ C(u) - R(u) + \sum_{m=0}^{n-1} \left(C\left(t\wedge T\cdot\left\lfloor \frac{t}{T}-m\right\rfloor\right) - C\left(T\cdot\left\lfloor \frac{t}{T}-m-1\right\rfloor\right) - Q\right) \vee 0 \right\}.$$

In figures 4 we have depicted an example of a task $R(t) = 3\cdot\lceil\frac{t}{3}\rceil \vee 0$, serviced by a resource $C(t) = 2\cdot t \vee 0$, on a deferrable server with $Q = 2$ and $T = 2$. Note, that as before, $C'(t)$ is flat whenever $R'(t)$ rises, and vice versa, but the pattern is different from the normal RTC processing.

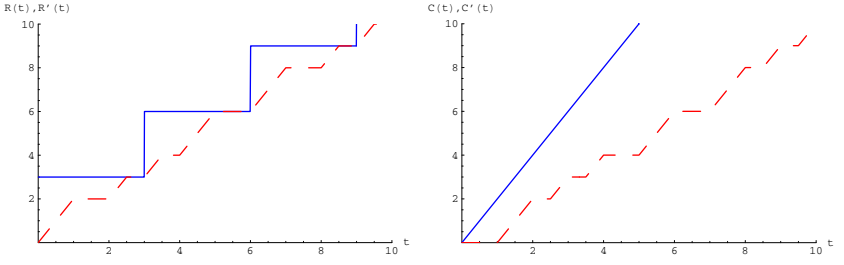


Fig. 4. Requests and resources in a deferrable server, (inputs straight, outputs dashed)

4 Scheduling Theorem

The delay of a task entering at time t is the time between its request and its completion. If we assume that *tasks are completed in the order in which they arrive*, the delay is the earliest time τ at which $R'(t+\tau) \ge R(t)$. The maximum delay Δ in the processing of a sequence of tasks is then defined by:

$$\Delta \triangleq \sup_t \inf\{\tau \mid R'(t) \ge R(t-\tau)\}.$$

Based on this definition, and the model of a deferrable server found in the previous section, we will now prove the following schedulability theorem. This theorem roughly states that task with a deadline greater than the sum of the minimum interarrival time of the task and the maximum delay between arrival of resources (due to other servers in the network) is schedulable (i.e. makes its deadline) provided that the tasks utilization is smaller than the utilization of the server and smaller than the utilization of the arriving resources.

Theorem 1 (Schedulability of a deferrable server). *Consider a deferrable server with period T and capacity Q. Assume that there is an upper bound S on the arrival time of tasks $R(t)$ such that for all $s \in \mathbb{R}$:*

$$\frac{R(s+S)-R(s)}{S} \leq \frac{Q}{T}.$$

Furthermore, assume that there is a lower bound $U \leq T$ on the arrival times of resources, such that for all $u \in \mathbb{R}$:

$$C(u+T)-C(u) \geq Q,$$
$$\frac{C(u+U)-C(u)}{U} \geq \frac{R(s+S)-R(s)}{S}.$$

Then, all relative deadlines of at least $S+2\cdot U$ are met, i.e.

$$\Delta \triangleq \sup_t \inf\{\tau \mid R'(t) \geq R(t-\tau)\} \leq S+2\cdot U.$$

Proof. The complete proof is by algebraic manipulation of the equations of the deferrable server model and can be found in the appendix.

In the special but in practice not uncommon case (see [1]) where the deferrable server has highest priority, the resource stream $C(t)$ can be considered to be linear, i.e. $C(t) = C \cdot t$. As a corollary, we then find that deadlines of at least S are met.

Corollary 1. *Consider a highest-priority deferrable server with period T and capacity Q. Furthermore, assume that we have an upper bound S on the arrival time of tasks $R(t)$ and a linear resource stream $C(t) = C \cdot t$, such that the respective utilizations satisfy the following inequalities for all $s \in \mathbb{R}$:*

$$C \geq \frac{Q}{T} \geq \frac{R(s+S)-R(s)}{S}.$$

Then, all relative deadlines of at least S are met, i.e.

$$\Delta \triangleq \sup_t \inf\{\tau \mid R'(t) \geq R(t-\tau)\} \leq S.$$

Proof. By assumption, we have for all $s \in \mathbb{R}$ that $\frac{R(s+S)-R(s)}{S} \leq C$. From linearity, it follows that for all $U > 0$ and all $u \in \mathbb{R}$ we have $\frac{C(u+U)-C(u)}{U} = C$, so $\frac{R(s+S)-R(s)}{S} \leq \frac{C(u+U)-C(u)}{U}$ and $C(u+T)-C(u) \geq Q$. Using our main theorem, we find for all $U > 0$, that relative deadlines greater than $S+2\cdot U$ are met, and hence $\Delta \triangleq \sup_t \inf\{\tau \mid R'(t) \geq R(t-\tau)\} \leq \inf_{U>0}\{S+2\cdot U\} = S$. Which concludes our proof.

5 Discussion

As mentioned in the introduction, our reason for making a formal model of the deferrable server strategy, was our suspicion that the schedulability analysis in [2] for real-time tasks under hierarchical fixed-priority preemptive scheduling and

a deferrable server, is in general pessimistic rather than exact. In this section, we will discuss this claim in more detail, with an apology for the necessary cluttering in use of terminology. Throughout the paper we have used the terminology that is usual in real-time calculus as much as possible, but in this section we must transliterate some of the results to fit the terminology of [2].

In [2], a comparison is made between deferrable servers and *periodic servers*, which only differ from each other in that a periodic server does not preserve its remaining capacity if resources are provided but tasks are not ready to use them, while a deferrable server does. The comparison is carried out under the assumption that there is a lowest priority, soft real-time task present (block 2 in figure 3), of which the interarrival time is unknown. Davis and Burns show that, under this assumption, the worst-case schedulability of tasks is better for periodic servers. However, we feel their subsequent conclusion that periodic servers dominate deferrable servers is somewhat misleading, because the presence of the soft real-time task severely influences the behavior of the deferrable server, while it does not influence the behavior of the periodic server. More precisely, in the worst case scenario, the buffer of the soft real-time task is never empty. This causes the deferrable server to loose capacity to this task at all times, and in effect behave in the same way as the periodic server.

Using Corollary 1, we will show that a deferrable server can indeed outperform a periodic server when this unknown soft real-time task is not present. The simplest example that shows this, is a system consisting of a single budgeted task, with a deadline equal to its period. In the remainder of this section, we first briefly relate our terminology with the terminology used in [2], and subsequently transliterate and refine Corollary 1 for our example system. Next, we recapitulate worst-case response time analysis given in [2] by presenting a dedicated equation for our special case. Under the aforementioned conditions, the analysis in [2] is exact for deferrable servers, i.e. provides a necessary and sufficient schedulability condition for the task. However, our corollary shows that the analysis is pessimistic for deferrable servers, when the unknown lowest priority task is not present or its interarrival time becomes known. This we illustrate using the aforementioned example.

In [2], a periodic task τ is characterized by a *period* (or *inter-arrival time*) T^τ, a *worst-case computation time* C^τ, and a *relative deadline* D^τ. We assume that the task's period and deadline are equal, i.e. $T^\tau = D^\tau$. A server σ is characterized by a *replenishment period* T^σ and a *capacity* C^σ. Based on these notions, the utilization U^τ of the task is given by $\frac{C^\tau}{T^\tau}$ and the utilization U^σ of the server by $\frac{C^\sigma}{T^\sigma}$.

The task τ can be either *bound* or *unbound*. The task τ is bound if it has a period that is an exact multiple of the server's period and an arrival time that coincides with the replenishment of the server's capacity. Otherwise τ is unbound. We assume an unbound task. Without loss of generality, we assume that the server σ is replenished for the first time at time $\varphi^\sigma = 0$. Moreover, we assume that τ is released for the first time at time $\varphi^\tau \geq 0$, i.e. *at* or *after* the first replenishment of σ. With this terminology in place, we can write

$$
\begin{aligned}
R(t) &= C^\tau \cdot \left\lceil \frac{t-\varphi^\tau}{T^\tau} \right\rceil \\
C(t) &= t \\
Q &= C^\sigma \\
S &= T^\tau \\
T &= T^\sigma
\end{aligned}
$$

For our system, we can now transliterate and refine Corollary 1.

Corollary 2. *Consider a highest-priority deferrable server σ with period T^σ and capacity C^σ. Furthermore, assume that the server is associated with a periodic task τ with period T^τ and worst-case computation time C^τ, where the first release of τ takes place at or after the first replenishment of σ. When the respective utilizations satisfy the following inequality*

$$U^\tau \leq U^\sigma \leq 1,$$

the deadline $D^\tau = T^\tau$ of τ is met.

Note that our transliterated corollary holds for both a bound task and an unbound task. Furthermore, note that (2) is a necessary and sufficient (i.e. exact) schedulability condition for both the task and the server. Finally, note that

$$U^{\mathcal{T}} \leq U^\sigma \leq 1,$$

is a *necessary* schedulability condition for a set $\mathcal{T}$ of independent hard real-time tasks with utilization $U^{\mathcal{T}}$ with an associated server σ with utilization U^σ

We will now derive a schedulability condition for our system from [2] starting from an equation to determine the task's worst-case response time. The task's *worst-case response time* WR^τ is the longest possible time from its arrival to its completion. Similarly, the server's *worst-case response time* WR^σ is the longest possible time from the server being replenished to its capacity being exhausted, given the task is ready to use all of its capacity. The task is said to be *schedulable* if (and only if)

$$WR^\tau \leq D^\tau.$$

Similarly, the server is schedulable if (and only if) $WR^\sigma \leq T^\sigma$.

For our system, the server is schedulable when $C^\sigma \leq T^\sigma$. Based on [2], we derive for our system that WR^τ is given by

$$WR^\tau = C^\tau + \left\lceil \frac{C^\tau}{C^\sigma} \right\rceil (T^\sigma - C^\sigma), \tag{1}$$

which leads to the following condition for schedulability of a task with a deadline D^τ equal to its period T^τ

$$C^\tau + \left\lceil \frac{C^\tau}{C^\sigma} \right\rceil (T^\sigma - C^\sigma) \leq T^\tau. \tag{2}$$

Now, as an example, we fix C^τ and T^τ and plot the minimum utilization U^σ_{min} of the server as a function of T^σ, i.e. we plot

$$U^\sigma_{min}(T^\sigma) = \min \left\{ \frac{C^\sigma}{T^\sigma} \mid T^\tau \geq C^\tau + \left\lceil \frac{C^\tau}{C^\sigma} \right\rceil (T^\sigma - C^\sigma), C^\sigma \geq 0 \right\}.$$

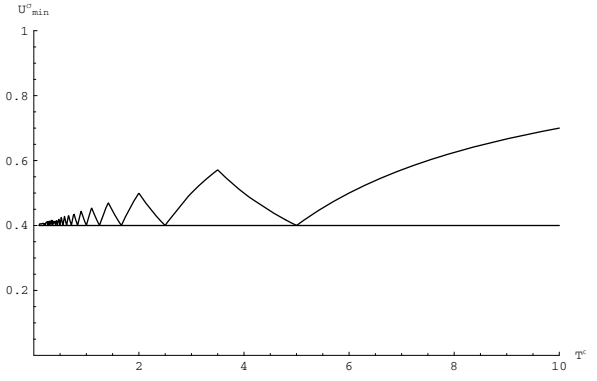


Fig. 5. Minimum server utilization U^{σ}_{min} for $C^{\tau} = 2$ and $T^{\tau} = 5$ as a function of T^{σ}

The result for $C^{\tau} = 2$ and $T^{\tau} = 5$ is depicted in figure 5. The horizontal line in this figure, shows the utilization $U^{\tau} = \frac{C^{\tau}}{T^{\tau}}$ of the task.

The figure illustrates that according to [2] only for values of T^{σ} equal to an integral fraction of T^{τ}, i.e. $T^{\sigma} = \frac{T^{\tau}}{n}$ for $n \in \mathbb{N}^{+}$, the minimum server utilization U^{σ}_{min} needed for schedulability is equal to the task utilization U^{τ}. For other values of T^{σ}, U^{σ}_{min} is higher than U^{τ}. However, according to our theorem, the server utilization U^{σ} may be chosen equal to the task utilization U^{τ} when using a deferrable server, irrespective of T^{σ}. From this example, we conclude that the schedulability condition expressed by (2) is sufficient but not necessary for an unbound task with an associated deferrable server. As a result, equation (1) is pessimistic, and the worst-case response time analysis presented in [2] for unbound tasks with associated deferrable servers is therefore pessimistic, and not exact, when the assumption of an unknown soft real-time lowest priority task is dropped.

According to [2], their worst-case response time analysis is exact for both deferrable servers and periodic servers. Based on that result, they claim that periodic servers dominate deferrable servers with respect to schedulability of tasks, i.e. "*there are no systems ... that can be scheduled using a set of deferrable servers that cannot also be scheduled using an equivalent set of periodic servers*". We have shown here, that this claim heavily relies on the assumed presence of an *unknown* soft real-time lowest priority task, and that it may not hold if such a task is not present, or if we somehow have more information on the interarrival time of this lowest priority task.

6 Conclusive Remarks and Future Work

We presented a formal model of a deferrable server budgeting strategy [10], using the real-time calculus of [11]. Using this model we derived a schedulability theorem stating that a set of tasks with deadlines greater than the sum of the minimum interarrival times of the tasks and the maximum delay between arrival of resources (due to other servers in the network) is schedulable provided that the

utilization of the tasks is smaller than the utilization of the server and smaller than the utilization of the arriving resources. Application of this theorem to the special case of a highest-priority server has led to an example where deferrable servers outperform periodic servers in the absence of soft real-time tasks. Hence, the claim in [2] that periodic servers outperform deferrable servers, holds in the presence of soft real-time tasks, but not in their absence. Worst-case response time analysis of deferrable servers therefore still requires further research.

In this paper, we used real-time calculus as an aid to prove our schedulability theorem for deferrable servers. In particular, we have proved the schedulability theorem using direct symbolic manipulation of our model of the deferrable server. Another, indirect way to analyse the behaviour of real-time calculus models, is through so-called *service curves* [4], comparable to analysis using Fourier and Laplace transformations in system theory. Service curve transformations define a system in terms of upper and lower bounds on the input and output streams rather than defining the streams exactly. Initial investigations suggest that service curve transformations can also be found for deferrable servers, but will probably not be *tight*, i.e. they will not give optimal bounds. Our attempts to use service curve transformations to prove our schedulability theorem failed, because the exact timing of replenishments turned out to be crucial in this proof, and is lost during the transformations.

Finally, the treatment of budgeting servers in general does not stop with the treatment of deferrable servers. First of all, there is a great variety of budgeting servers already in use in practice and theory. And secondly, the use of budgeting servers has lead to the introduction of a hierarchical approach to scheduling. If multiple tasks are run on a single server, one may first divide the available resources over the servers, and in a second stage schedule the tasks that run on each server independently of what runs on the other servers. Recent theory about the schedulability of tasks under different kinds of budgeting servers was presented in [2,3,7,9]. In the beginning of section 3 we have shown that there is a connection between the modeling of individual tasks and the modeling of servers. Of course, this concept can be lifted to meta-servers, etc. But, further analysis is still needed to obtain a truly hierarchical approach, in which we first abstract from lower level tasks and analyse the connections between servers, and afterwards analyse the properties of the served tasks.

Acknowledgements. We would like to thank Lothar Thiele of ETH Zürich, and Robert I. Davis and Alan Burns of the University of York, for their valuable feedback on our initial results and for helping us to better understand their respective work.

References

1. Buttazzo, G.C.: Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Springer Science+Business Media, Inc. (2005)
2. Davis, R.I., Burns, A.: Hierarchical fixed priority pre-emptive scheduling. In: Proc. 26^{th} IEEE Real-Time Systems Symposium (RTSS), pp. 389–398 (December 2005)

3. Deng, Z., Liu, J.W.-S.: Scheduling real-time applications in open environment. In: Proc. 18^{th} IEEE Real-Time Systems Symposium (RTSS), pp. 308–319 (December 1997)
4. Thiran, P., Le Boudec, J.-Y. (eds.): Network Calculus. LNCS, vol. 2050. Springer, Berlin (2001)
5. Joseph, M. (ed.): Real-Time Systems: Specification, Verification and Analysis. Prentice-Hall, Englewood Cliffs (1996)
6. Klein, M.H., Ralya, T., Pollak, B., Obenza, R., González-Harbour, M.: A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems. Kluwer Academic Publishers, Dordrecht (1993)
7. Kuo, T.-W., Li, C.-H.: A fixed-priority-driven open environment for real-time applications. In: Proc. 20^{th} IEEE Real-Time Systems Symposium (RTSS), pp. 256–267 (December 1999)
8. Rajkumar, R., Juvva, K., Molano, A., Oikawa, S.: Resource kernels: A resource-centric approach to real-time and multimedia systems. In: Proc. SPIE, Conference on Multimedia Computing and Networking (CMCN), vol. 3310, pp. 150–164 (January 1998)
9. Saewong, S., Rajkumar, R., Lehocky, J.P., Klein, M.H.: Analysis of hierarchical fixed-priority scheduling. In: Proc. 14^{th} Euromicro Conference on Real-Time Systems (ECRTS), pp. 152–160 (2002)
10. Strosnider, J.K., Lehoczky, J.P., Sha, L.: The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. IEEE Transactions on Computers 44(1), 73–91 (1995)
11. Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: Proc. IEEE International Symposium on Circuits and Systems (ISCAS), vol. 4, pp. 101–104 (2000)

A Scheduling Theorem: Proof

To prove $\sup_t \inf\{\tau \mid R'(t) \geq R(t-\tau)\} \leq S + 2 \cdot U$, it is necessary and sufficient to prove that $R'(t) \geq R(t - S - 2 \cdot U)$ for all t. For this, we start from our previously obtained solution for $R'(t)$.

$$R'(t) = \inf_{n \in \mathbb{N}} \inf_{u \leq t \wedge T \cdot \left\lceil \frac{t}{T} - n \right\rceil} \left\{ R(u) + C\left(t \wedge T \cdot \left\lceil \frac{t}{T} - n \right\rceil\right) - C(u) + \sum_{m=0}^{n-1} Q \wedge \left(C\left(t \wedge T \cdot \left\lceil \frac{t}{T} - m \right\rceil\right) - C\left(T \cdot \left\lceil \frac{t}{T} - m - 1 \right\rceil\right)\right)\right\}$$

We split off the special cases where $n = 0$ and $m = 0$, and find:

$$\begin{aligned} &= \inf_{u \leq t} \{R(u) + C(t) - C(u)\} \\ &\wedge \inf_{n \geq 1} \inf_{u \leq T \cdot \left\lceil \frac{t}{T} - n \right\rceil} \left\{ R(u) + C\left(T \cdot \left\lceil \frac{t}{T} - n \right\rceil\right) - C(u) \right. \\ &+ Q \wedge \left(C(t) - C\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right)\right) \\ &\left. + \sum_{m=1}^{n-1} Q \wedge \left(C\left(T \cdot \left\lceil \frac{t}{T} - m \right\rceil\right) - C\left(T \cdot \left\lceil \frac{t}{T} - m - 1 \right\rceil\right)\right)\right\} \end{aligned}$$

Again, we find two cases, depending on whether Q or $C\left(t\right) - C\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil\right)$ is larger.

$$
\begin{aligned}
&= \inf_{u \le t} \left\{ R\left(u\right) + C\left(t\right) - C\left(u\right) \right\} \\
&\wedge \inf_{n \ge 1} \inf_{u \le T \cdot \left\lceil \frac{t}{T} - n \right\rceil} \left\{ R\left(u\right) + C\left(T \cdot \left\lceil \frac{t}{T} - n \right\rceil \right) - C\left(u\right) \right. \\
&\quad \left. + Q + \sum_{m=1}^{n-1} Q \wedge \left(C\left(t \wedge T \cdot \left\lceil \frac{t}{T} - m \right\rceil \right) - C\left(T \cdot \left\lceil \frac{t}{T} - m - 1 \right\rceil \right) \right) \right\} \\
\wedge \inf_{n \ge 1} \inf_{u \le T \cdot \left\lceil \frac{t}{T} - n \right\rceil} &\left\{ R\left(u\right) + C\left(t\right) - C\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil \right) + C\left(T \cdot \left\lceil \frac{t}{T} - n \right\rceil \right) - C\left(u\right) \right. \\
&\left. + \sum_{m=1}^{n-1} Q \wedge \left(C\left(T \cdot \left\lceil \frac{t}{T} - m \right\rceil \right) - C\left(T \cdot \left\lceil \frac{t}{T} - m - 1 \right\rceil \right) \right) \right\}
\end{aligned}
$$

Take $u = T \cdot \left\lceil \frac{t}{T} - m - 1 \right\rceil$ in the assumption on resource arrivals to find $C\left(T \cdot \left\lceil \frac{t}{T} - m \right\rceil\right) - C\left(T \cdot \left\lceil \frac{t}{T} - m - 1 \right\rceil\right) \ge Q$.

$$
\begin{aligned}
&= \inf_{u \le t} \left\{ R\left(u\right) + C\left(t\right) - C\left(u\right) \right\} \\
&\wedge \inf_{n \ge 1} \inf_{u \le T \cdot \left\lceil \frac{t}{T} - n \right\rceil} \left\{ R\left(u\right) + C\left(T \cdot \left\lceil \frac{t}{T} - n \right\rceil \right) - C\left(u\right) + n \cdot Q \right\} \\
&\wedge \inf_{n \ge 1} \inf_{u \le T \cdot \left\lceil \frac{t}{T} - n \right\rceil} \left\{ R\left(u\right) + C\left(t\right) - C\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil \right) \right. \\
&\quad \left. + C\left(T \cdot \left\lceil \frac{t}{T} - n \right\rceil \right) - C\left(u\right) + \left(n-1\right) \cdot Q \right\}
\end{aligned}
$$

Then, we truncate the argument of $C\left(.\right)$ to a multiple of U with a convenient remainder. Using monotonicity of $C\left(t\right)$ we find:

$$
\begin{aligned}
&\ge \inf_{u \le t} \left\{ R\left(u\right) + C\left(\left\lfloor \frac{t-u}{U} \right\rfloor \cdot U + u \right) - C\left(u\right) \right\} \\
&\wedge \inf_{n \ge 1} \inf_{u \le T \cdot \left\lceil \frac{t}{T} - n \right\rceil} \left\{ R\left(u\right) + C\left(\left\lfloor \frac{T \cdot \left\lceil \frac{t}{T} - n \right\rceil - u}{U} \right\rfloor \cdot U + u \right) - C\left(u\right) + n \cdot Q \right\} \\
&\wedge \inf_{n \ge 1} \inf_{u \le T \cdot \left\lceil \frac{t}{T} - n \right\rceil} \left\{ R\left(u\right) + C\left(\left\lfloor \frac{t - T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil}{U} \right\rfloor \cdot U + T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil \right) \right. \\
&\quad \left. - C\left(T \cdot \left\lceil \frac{t}{T} - 1 \right\rceil \right) + C\left(\left\lfloor \frac{T \cdot \left\lceil \frac{t}{T} - n \right\rceil - u}{U} \right\rfloor \cdot U + u \right) - C\left(u\right) + \left(n-1\right) \cdot Q \right\}
\end{aligned}
$$

Now, we define X as a lower bound on the utilization of $R\left(t\right)$, and find:

$$
\inf_{u} \left\{ \frac{C\left(u+U\right) - C\left(u\right)}{U} \right\} \ge \sup_{s} \left\{ \frac{R\left(s+S\right) - R\left(s\right)}{S} \right\} \triangleq X.
$$

Using X, we eliminate $C(t)$ and Q.

$$
\begin{aligned}
&\geq \inf_{u \leq t} \left\{ R(u) + \left\lfloor \frac{t-u}{U} \right\rfloor \cdot U \cdot X \right\} \\
&\wedge \inf_{n \geq 1} \inf_{u \leq T \cdot \lceil \frac{t}{T} - n \rceil} \left\{ R(u) + \left\lfloor \frac{T \cdot \lceil \frac{t}{T} - n \rceil - u}{U} \right\rfloor \cdot U \cdot X + n \cdot T \cdot X \right\} \\
&\wedge \inf_{n \geq 1} \inf_{u \leq T \cdot \lceil \frac{t}{T} - n \rceil} \left\{ R(u) + \left\lfloor \frac{t - T \cdot \lceil \frac{t}{T} - 1 \rceil}{U} \right\rfloor \cdot U \cdot X \right. \\
&\quad \left. + \left\lfloor \frac{T \cdot \lceil \frac{t}{T} - n \rceil - u}{U} \right\rfloor \cdot U \cdot X + (n-1) \cdot T \cdot X \right\}
\end{aligned}
$$

Observe that for all x we have $x \cdot X \geq \lfloor \frac{x}{S} \rfloor \cdot S \cdot X$. And since $S \cdot X$ serves as an upper bound on $R(s+S) - R(s)$, this gives us

$$
\begin{aligned}
&\geq \inf_{u \leq t} \left\{ R\left(u + \left\lfloor \left\lfloor \frac{t-u}{U} \right\rfloor \cdot \frac{U}{S} \right\rfloor \cdot S \right) \right\} \\
&\wedge \inf_{n \geq 1} \inf_{u \leq T \cdot \lceil \frac{t}{T} - n \rceil} \left\{ R\left(u + \left\lfloor \left\lfloor \frac{T \cdot \lceil \frac{t}{T} - n \rceil - u}{U} \right\rfloor \cdot \frac{U}{S} + n \cdot \frac{T}{S} \right\rfloor \cdot S \right) \right\} \\
&\wedge \inf_{n \geq 1} \inf_{u \leq T \cdot \lceil \frac{t}{T} - n \rceil} \\
&\quad \left\{ R\left(u + \left\lfloor \left\lfloor \frac{t - T \cdot \lceil \frac{t}{T} - 1 \rceil}{U} \right\rfloor \cdot \frac{U}{S} + \left\lfloor \frac{T \cdot \lceil \frac{t}{T} - n \rceil - u}{U} \right\rfloor \cdot \frac{U}{S} + (n-1) \cdot \frac{T}{S} \right\rfloor \cdot S \right) \right\}
\end{aligned}
$$

And using monotonicity of $R(t)$, together with the observation that $\lfloor x \rfloor \cdot y \geq x \cdot y - y$ we find:

$$
\begin{aligned}
&\geq \inf_{u \leq t} \left\{ R\left(u + \left\lfloor \frac{t-u}{U} \right\rfloor \cdot U - S \right) \right\} \\
&\wedge \inf_{n \geq 1} \inf_{u \leq T \cdot \lceil \frac{t}{T} - n \rceil} \left\{ R\left(u + \left\lfloor \frac{T \cdot \lceil \frac{t}{T} - n \rceil - u}{U} \right\rfloor \cdot U + n \cdot T - S \right) \right\} \\
&\wedge \inf_{n \geq 1} \inf_{u \leq T \cdot \lceil \frac{t}{T} - n \rceil} \\
&\quad \left\{ R\left(u + \left\lfloor \frac{t - T \cdot \lceil \frac{t}{T} - 1 \rceil}{U} \right\rfloor \cdot U + \left\lfloor \frac{T \cdot \lceil \frac{t}{T} - n \rceil - u}{U} \right\rfloor \cdot U + (n-1) \cdot T - S \right) \right\} \\
&\geq \inf_{u \leq t} \{ R(u + t - u - U - S) \} \\
&\wedge \inf_{n \geq 1} \inf_{u \leq T \cdot \lceil \frac{t}{T} - n \rceil} \left\{ R\left(u + T \cdot \left\lfloor \frac{t}{T} - n \right\rfloor - u - U + n \cdot T - S \right) \right\} \\
&\wedge \inf_{n \geq 1} \inf_{u \leq T \cdot \lceil \frac{t}{T} - n \rceil}
\end{aligned}
$$

$$
\begin{aligned}
&\left\{R\left(u+t-T\cdot\left\lceil\frac{t}{T}-1\right\rceil-U+T\cdot\left\lceil\frac{t}{T}-n\right\rceil-u-U+(n-1)\cdot T-S\right)\right\}\\
&=R\left(t-U-S\right)\\
&\wedge\ R\left(T\cdot\left\lceil\frac{t}{T}\right\rceil-U-S\right)\\
&\wedge\ R\left(t-2\cdot U-S\right)\\
&=R\left(t-2\cdot U-S\right)
\end{aligned}
$$

Which concludes our proof.

Automatic Abstraction Refinement for Timed Automata*

Henning Dierks[1], Sebastian Kupferschmid[2], and Kim G. Larsen[3]

[1] OFFIS, Oldenburg, Germany
dierks@offis.de
[2] University of Freiburg, Germany
kupfersc@informatik.uni-freiburg.de
[3] Aalborg University, Denmark
kgl@cs.auc.dk

Abstract. We present a fully automatic approach for counterexample guided abstraction refinement of real-time systems modelled in a subset of timed automata. Our approach is implemented in the MOBY/RT tool environment, which is a CASE tool for embedded system specifications. Verification in MOBY/RT is done by constructing abstractions of the semantics in terms of timed automata which are fed into the model checker UPPAAL. Since the abstractions are over-approximations, absence of abstract counterexamples implies a valid result for the full model. Our new approach deals with the situation in which an abstract counterexample is found by UPPAAL. The generated abstract counterexample is used to construct either a concrete counterexample for the full model or to identify a slightly refined abstraction in which the found spurious counterexample cannot occur anymore. Hence, the approach allows for a fully automatic abstraction refinement loop starting from the coarsest abstraction towards an abstraction for which a valid verification result is found. Nontrivial case studies demonstrate that this approach computes small abstractions fast without any user interaction.

1 Introduction

Embedded systems often control safety critical systems. Hence, formal methods are mandatory to establish correctness results for such systems. Since model checking is a technique that does the analysis without any user interaction it is widely examined in the literature and many tools are available. However, model checking suffers from the so called state space explosion problem, i. e., the complexity of the verification procedure is exponential in the size of the system. Therefore, several techniques have been invented to tackle this problem like symbolic representation and abstractions. In most cases the requirements of embedded systems refer to the timing. Verification of real-time systems by model checking is even more difficult because time adds another source of complexity. As a consequence, checking a model that represents a nontrivial real-time system always requires to find an appropriate abstraction that can be checked

* 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). See http://www.avacs.org/ for more information.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 114–129, 2007.

within the given resources of memory and time. If the used abstraction is an over-approximation, then the absence of an abstract counterexample implies the absence of concrete counterexamples. Such an abstraction is called a *safe* abstraction. However, if a model checker analyses a safe abstraction and discovers a counterexample, the question arises whether the counterexample is spurious or not, i. e., whether there is a concrete counterpart in the full model.

Without automation, model checking by abstractions consists of several, error-prone and time-consuming tasks. First, the user has to select a proper initial abstraction which usually requires a deep knowledge of the system under consideration. If the model checker finds a counter example in this abstraction, then the user has to analyse whether it is spurious or not. If the counterexample is real, then the user is done, otherwise the selected abstraction was too coarse. In this case the initial abstraction must be refined so that the spurious counterexample is eliminated. These steps are repeated until there is no abstract counterexample, or the abstract counterexample is also a counterexample for the full system.

The approach presented in this paper automates *all* steps of the abstraction refinement loop in the setting of real-time specifications given as PLC automata [1]. This leads to a fully automated abstraction refinement loop as depicted in Fig. 1. This loop starts with the coarsest possible abstraction and iterates as long as spurious counterexamples are found. If the model checker finds an abstract counterexample, then a counterexample analyser is used to check whether it is spurious. Our analyser first constructs a test automaton from the abstract counterexample. The composition of the full model with the newly generated automaton is then fed into the model checker. We extended the model checker in such a way that if the counterexample is spurious, the model checker also reports hints why it is spurious. These hints are then used in the refinement step in order to eliminate the abstract counterexample. The termination of this abstraction refinement loop is guaranteed by the fact that each iteration removes at least one of finitely many abstracted variables.

To demonstrate the potential of our approach we carried out several nontrivial case studies, i. e., the respective full models cannot be handled within the given memory resources. Our approach is able to handle them with only a fraction of the resources, starting from the initial abstraction towards a refined abstraction for which a definite answer could be found.

The remainder of the paper is structured as follows: the next section briefly introduces the formalisms we work with, Section 3 shows how we check for spuriousness

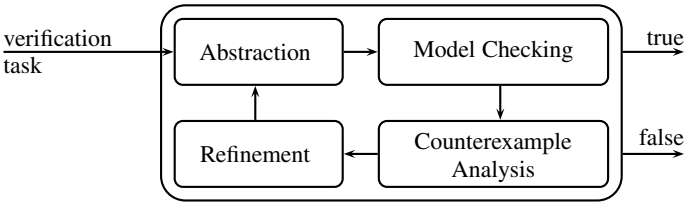


Fig. 1. Counterexample guided abstraction refinement loop

of counterexamples. Section 4 describes our used abstraction refinement cycle and the algorithms to provide the necessary information. Section 5 provides experimental evaluation of the implementation of the method within UPPAAL. In Section 6 we discuss related work and Section 7 concludes.

2 Preliminaries

In our setting, a verification task consists of a real-time system, which is given in terms of PLC automata, together with a temporal property to verify. To solve such a verification task with our framework, we use MOBY/RT to compute an abstraction of the given PLC automata. MOBY/RT is a tool for the development and analysis of PLC automata [2]. From the verification perspective the most important feature of MOBY/RT is the generation of safe abstractions of an arbitrary set of PLC automata together with a temporal property into the input syntax of UPPAAL 3.4. These abstractions are generated according to the entities (e. g., variables and delays) of the PLC automata the user has chosen for abstraction.

In the following we will briefly describe PLC automata, timed automata and the relation between these two formalisms.

2.1 PLC Automata

The formal specification language called PLC automata has been developed to enable formal verification of real-time properties of PLC programs. A Programmable Logic Controller (PLC) is a standardised hardware platform which is especially equipped to simplify the design of real-time controllers in practice. It can be seen as a simple computer with a special real-time operating system. A PLC communicates with the environment via unbuffered asynchronous input and output channels. The environment may change the values of the inputs arbitrarily whereas the outputs are controlled by the PLC. PLCs behave in a cyclic manner where every cycle consists of the following three phases: first the inputs are polled, then the new output values are computed and finally the outputs are updated. The repeated execution of this cycle is managed by the operating system. The only part the programmer has to adapt is the computing phase. Depending on the program and on the number of inputs and outputs there is an upper time bound for such a cycle.

In the definition of PLC automata we consider the upper time bound for a complete cycle and the possibility to delay the system's reactions depending on state and input. Figure 2 gives an example of a PLC automaton. The automaton has three locations q_0, q_1, q_2 and an output variable `output` that ranges over `ok`, `test` and `alarm`. It reacts to a Boolean input variable `signal`. Every state has two labels shown below its name in the picture. They define a delay time d and a constraint S on the input. The value of d defines the *minimal* amount of time that the system should stay in the corresponding state provided that, meanwhile, only input values satisfying S are polled.

A PLC automaton describes the behaviour of the system in the computation phase. The operational behaviour is similar to a finite state machine, i. e., depending on the polled input value the system changes both its state and its output. The behaviour is

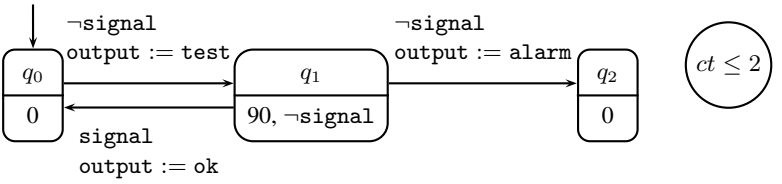


Fig. 2. An example of a PLC automaton

modified in only one case: if the annotations of the current state are d and S, then a transition is only executed when the polled input does not satisfy S *or* the current state holds longer than d time units. That means a transition is disabled if the polled input satisfies S and the current state has not exceeded the delay time d.

Thus, the PLC automaton in Fig. 2 behaves as follows: it starts in state q_0 and remains there as long as it reads only the input `signal`. The first time it reads `¬signal` it changes to state q_1. In q_1 the automaton reacts to the input value `signal` by changing the state back to q_0 *independently* of the time it stayed in state q_1. If it reads `¬signal` in q_1 the behaviour of the system depends on the duration it already stayed in q_1. If 90 time units have elapsed it can take the transition to q_2. Otherwise, no transition is fired. In q_2 the automaton remains forever. Hence, we know that the automaton changes its output to `alarm` when `¬signal` holds a little bit longer than 90 time units because the cycle time (2 time units) has to be considered.

Note that PLC automata are implementable. In [3] a translation of PLC automata into source code for PLCs is given. In [2] also a translation into C++ code (tailored to Lego Mindstorms) has been developed. The intended logical relationship between the execution of the code by the real-world hardware and the semantics given in the rest of this paper is *refinement*. In other words: the real-world implementation cannot show a behaviour that is not covered by the formal semantics.

2.2 UPPAAL and Timed Automata

UPPAAL[1] is a modelling, simulation, and verification tool for real-time systems modelled as networks of extended timed automata [4,5]. We assume the reader is roughly familiar with timed automata and their commonly used extensions. Therefore, we only provide a short description of timed automata and the extensions which we use for the construction of test automata. For more details about this tool the reader is referred to the UPPAAL tutorial [6].

Figure 3 shows a network of three parallel timed automata P, Q and R as it is used in UPPAAL. The system has three clocks x, y and z, three integer variables k, m and n, binary synchronisation labels a and b and a committed location q_2 (indicated by the "c:"). The initial state of the overall system is given by the three initial locations p_1, q_1 and r_1 together with the initial values of the integer variables and the initial values for the clocks. All clocks and integer values are 0 in this state. The bounded integer variables are part of each system state, they can occur in transition guards and can be changed with the assignment of a transition. Two edges of different automata can fire

[1] See `http://www.uppaal.com/`

a synchronised transition of the system if they are labelled with complementary synchronisation labels (represented by "!" and "?" respectively). Suppose the system is in a state where it is in p_2 and q_1 and the guards of the outgoing edges of these two locations are enabled, then the system can take a synchronised transition, which leads to a state where P is in p_3 and Q is in q_2. For an edge annotated with a synchronisation label it is not possible to fire a transition without a synchronisation partner. Edges without synchronisation label are called τ-transitions. A system state is a committed state if at least one of the current automata locations is committed. Committed locations are a mechanism to restrict the behaviour of the overall system. A committed state cannot delay and the next transition applied to this state must involve an outgoing edge of a committed location. For example, if the system is in a state where P is at p_1, Q is at the committed location q_2 and R is at r_1, then this system state is committed. The system then has to take the edge from q_2 to q_3 which requires Q to synchronise with R. The location p_1 is labelled with a location invariant. This restricts the duration the system can idle in the current state. Suppose the system is in a state where P is at p_1 and the value of the clock x is 0, then the system can remain at most 11 time units before leaving this state.

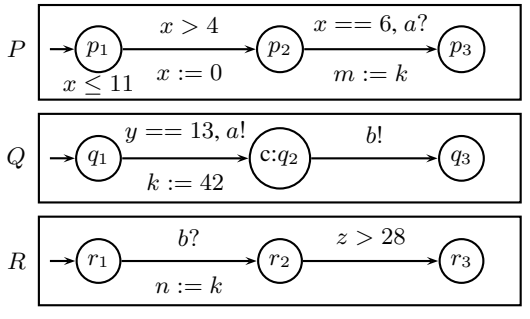


Fig. 3. A simple UPPAAL model

2.3 Timed Automata Semantics of PLC Automata

The semantics of PLC automata is defined in terms of timed automata [3, 1]. Due to space limitations we can just present a sketch using the example of Fig. 2. The semantics of this automaton, depicted in Fig. 4, consists of two timed automata and the following (global) variables and clocks: z is a clock that represents the duration of the PLC cycle, y is a clock that measures how long the system stays in q_1, `Out` and `sig` represent the variables `Output` resp. `signal` used in the PLC automaton and `Psig` is a variable that represents the *polled* value of the input variable `signal`. The states of the PLC automaton appear as a set of locations in the timed automaton. For example, the state q_0 has two representatives in the timed automaton in order to represent the internal state of the PLC within the cycle (q_0/p stands for polling, q_0/cu stands for computing and updating). The transitions between q_0/p and q_0/cu implement the polling behaviour of the PLC automaton in state q_0. The polling step copies the current value of `sig` to the variable `Psig` which is used for the subsequent computations. The outgoing transitions from q_0/cu represent the reactions of the PLC automaton in state q_0.

Depending on the polled value of `sig` the system switches to q_1 or remains in q_0. Moreover, these transitions reset the z clock because the cycle has been finished. If the automaton switches to q_1 the output variable is changed appropriately and the clock y is reset to be able to check whether 90 time units have elapsed while staying in state q_1. Since q_1 is equipped with a delay the semantics needs more locations to represent the behaviour. After polling (transition from q_1/p to q_1/c) the timed automaton checks whether the delay time has passed or the delay condition is not satisfied. If this is the case, then a transition to q_1/u is enabled. Otherwise the system can switch to q_1/d ("delayed") where the cycle is finished. Note that the semantics is nondeterministic with respect to the timing in order to model the physical reality. To ensure progress each state has the invariant $z \leq 2$. Therefore, the timed automaton has to execute a cycle within the upper bound because only transitions to q_i/p reset the z clock. To model the environment that can change the input variable `sig` arbitrarily a driver automaton is added that can always toggle the input.

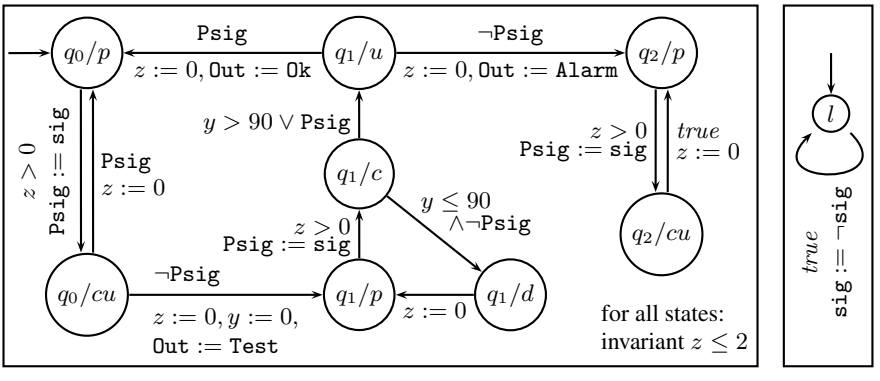


Fig. 4. Semantics of the PLC automaton from Fig. 2 in terms of timed automata

Abstractions of this semantics are generated by selecting a set of variables and clocks. Then the abstract semantics is derived from the full semantics by removing all assignments to the abstracted variables and clocks and replacing all constraints by the strongest constraint that is weaker than the original and that does not contain abstracted entities. For example, the guard $y \leq 90 \wedge$ `Psig` is replaced by `Psig` if the clock y is abstracted and `sig` is not.

3 Counterexample Analysis

In the abstraction refinement loop, when model checking a safety property of an abstract system returns a counterexample, one has to investigate whether it is spurious or not. One possible way of doing this is to build a *linear* test automaton T, in which every location has at most one outgoing edge, from the abstract counterexample. This test automaton is then composed with the full system F to be verified. If the last location of the test automaton is reachable in the composed system, then we know that the abstract counterexample is also a counterexample for the full system.

It is desirable that the test automaton T is able to find a concretisation if there is any and, moreover, it is mandatory that T restricts the behaviour of the full system as much as possible. If this is not the case, the test automaton is useless because it would make no difference to model check the full system only. We therefore construct the test automaton so that it synchronises with the full model as often as possible. By doing this, only a small fraction of the state space of the full model is explored. This makes perfect sense, because we are only interested in a concretisation of the abstract trace.

A UPPAAL counterexample is a finite sequence of states that are either connected via transitions or via delays with a certain duration. For example, for the timed automata system given in Fig. 3 and the temporal formula $\mathrm{E}\Diamond(p_3 \wedge q_3 \wedge r_3)$, UPPAAL reports the error trace given in Fig. 5. The trace starts with the initial state of the system and ends with a state satisfying the given property. Each state of the trace assigns each automaton of the system a location, each variable a value of the variable's domain and each clock a rational value. A test automaton built from such a trace proceeds if the full model of the system executes a transition that enables the guard of the next transition of the test automaton. The problem is to decide whether a step in the full model matches. Checking the values of the variables is straight-forward. To check the clock values is a bit tricky because a trace may also have clock valuations with rational numbers. Below we show how this problem can be solved. Another problem is to match the locations. The locations are not subject to abstractions. Hence, they appear in both the full and the abstract model. However, UPPAAL does not provide any syntactic means to refer to locations in guards. Therefore, we introduce an auxiliary integer variable for each automaton of the system. The current value of this variable identifies uniquely the current location of the automaton. Thus, the test automaton is able to match locations by checking the corresponding auxiliary variable. In order to preserve the same behaviour with respect to global time we add a clock *time* to the test automaton. This clock is never reset and therefore it represents the current duration of the trace at any time. The next three paragraphs give a detailed description on how to build a test automaton.

The first element of an abstract trace is the initial state of the abstract system. Although we do not expect any differences with the full model here, we add a corresponding check for reasons of completeness. We add a transition leading from the initial location t_0 of the test automaton to a new location t_1. The guard of this transition checks if the system's variables have the right values at time point $time = 0$. To restrict the full model's behaviour t_0 is a committed location.

A delay transition with duration $d \in \mathbb{Q}$ of the abstract counterexample is translated as follows: suppose that, after the transition is made, the system is in a state which is described by the valuation val and that $T \in \mathbb{Q}$ is the sum of all durations that occur before the transition. Let the most recently added location of the test automaton be t_n. In this case we add two locations t_{n+1} and t_{n+2} to the test automaton and the two transitions $t_n \xrightarrow{await(T+d)} t_{n+1}$ and $t_{n+1} \xrightarrow{await(T+d) \wedge check(val)} t_{n+2}$. Here $await(q)$ for $q \in \mathbb{N}$ is $time = q$ and for $q \in \mathbb{Q} \setminus \mathbb{N}$ is $\lfloor q \rfloor > time \wedge time < \lceil q \rceil$. Note that by the definition of $await(q)$ the test automaton searches for an over-approximation of the given abstract trace. The expression $check(val)$ is the conjunction of tests whether all discrete variables of val equal the valuation of the state. Note that this approach is correct because if there is no concretisation found using this over-approximation then there is no concretisation at

```
( P.p1 Q.q1 R.r1 ) x=0 y=0 z=0 k=0 m=0 n=0
   Delay: 7
( P.p1 Q.q1 R.r1 ) x=7 y=7 z=7 k=0 m=0 n=0
   P.p1->P.p2  x > 4, tau, x := 0
( P.p2 Q.q1 R.r1 ) x=0 y=7 z=7 k=0 m=0 n=0
   Delay: 6
( P.p2 Q.q1 R.r1 ) x=6 y=13 z=13 k=0 m=0 n=0
   Q.q1->Q.q2  y == 13, a!, k := 42
   P.p2->P.p3  x == 6, a?, m := k
( P.p3 Q.q2 R.r1 ) x=6 y=13 z=13 k=42 m=42 n=0
   Q.q2->Q.q3  1, b!, 1
   R.r1->R.r2  1, b?, n := k
( P.p3 Q.q3 R.r2 ) x=6 y=13 z=13 k=42 m=42 n=42
   Delay: 15.5
( P.p3 Q.q3 R.r2 ) x=21.5 y=28.5 z=28.5 k=42 m=42 n=42
   R.r2->R.r3  z > 28, tau, 1
( P.p3 Q.q3 R.r3 ) x=21.5 y=28.5 z=28.5 k=42 m=42 n=42
   Delay: 0.5
( P.p3 Q.q3 R.r3 ) x=22 y=29 z=29 k=42 m=42 n=42
```

Fig. 5. Trace reported by UPPAAL for the query $\mathrm{E}\Diamond(p_3 \wedge q_3 \wedge r_3)$ of the system from Fig. 3

all. If a concretisation is found then the concrete trace may differ from the abstract trace with respect to the non-integer delays but it is a trace of the full model that satisfies the reachability property. An alternative approach would be to multiply all timing constants with the common denominator and check for equality only.

A τ-transition in the counterexample also introduces two locations in the test automaton. Suppose the last added location of the test automaton is t_n and after the τ-transition the system is in a state described by the valuation val. Let T and t_n be defined as above. Here, we add the locations t_{n+1} and t_{n+2} to the test automaton and the two transitions $t_n \xrightarrow{await(T) \wedge C?} t_{n+1}$ and $t_{n+1} \xrightarrow{await(T) \wedge check(val)} t_{n+2}$. Note that we exploit the fact that the abstract model made a τ-transition to introduce a synchronisation of the test automaton with the full model via the channel C. By this, the test automaton is notified as soon as the full model changed the values for the variables and the search space is reduced. Since time is supposed to pass, the location t_n cannot be committed. However, to restrict the behaviour of the system t_{n+1} can be committed.

A synchronised transition of the counterexample is translated as follows: suppose again, that the valuation val, T and t_n are defined as above. In contrast to how we handle a τ-transition, we cannot force the test automaton to participate. However, we know that whenever MOBY/RT generates a synchronised transition this only happens directly after a τ-transition and that all synchronised transitions are leaving a committed location. Therefore we add the location t_{n+1} and the transition $t_n \xrightarrow{check(val) \wedge await(T)} t_{n+1}$. As synchronised transitions can happen sequentially when more than one automaton is in the network and the transition's origin is a committed location we have to make location t_n committed, too.

Consider the trace generated by UPPAAL in Fig. 5 again. If we apply the above construction rules we get the test automaton described in Fig. 6.

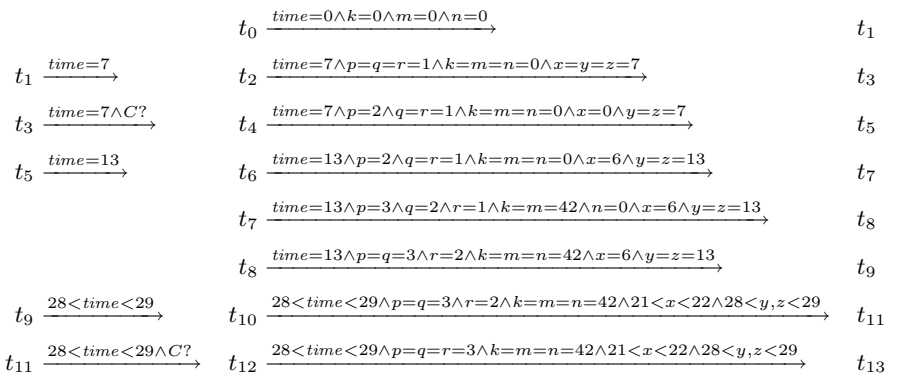


Fig. 6. Test automaton for the trace in Fig. 5

4 Abstraction Refinement

When an abstract counterexample reveals to be spurious, a model checker normally does not give any hint why this is the case. Usually, the next steps one has to do in order to get a refined version of the current abstraction is to analyse the counterexample. Here, one has to identify variables or clocks respectively that hinder the progress of the test automaton. Normally, this has to be done *manually* and, on the one hand, is a tedious and time consuming procedure and on the other hand requires a deep understanding of the model to verify. We will henceforth refer to integer variables and clocks as variables respectively.

Our approach automates the analysis of the counterexample. We did this by extending UPPAAL so that if an abstract counterexample is spurious, UPPAAL reports this *and* at the same time provides a set of variables that should not be abstracted in the next iteration of the abstraction refinement loop.

To determine a refined abstraction, we exploit the fact that our test automata are linear. If the abstract counterexample turns out to be spurious, then there is a unique transition in the test automaton whose starting location is reached, but not its target location. We call the starting location of this transition the *dead end location* and the transition itself we call the *dead end transition*. The dead end transition can either be blocked because there is no enabled transition in the full system that can synchronise with the test automaton, or when there is no reachable state with a valuation of the variables that satisfies the guard of the dead end transition.

Our approach determines a minimal set of variables u_{hint}, so that if these variables had different values, the test automaton could take at least the dead end transition. This is done on the fly, while UPPAAL checks if the error trace is spurious. Figure 7 sketches UPPAAL's verification algorithm for safety properties. The arguments of the *verify* function are the initial state of the system s_0 and the property ϕ to verify. We extended this algorithm by including the lines 13–15.

During the analysis, UPPAAL checks each outgoing transition from a location in the current state s if it is enabled. If this is the case, the successor state s' is computed

```
1  function verify(s_0, φ):
2    open = {s_0}, closed = ∅
3    while open ≠ ∅:
4      s = open.pop()
5      if s ⊨ φ:
6        return True
7      if s ∉ closed:
8        closed.push(s)
9        for each outgoing transition t of s:
10         if t is enabled:
11           s' = succ(s, t)
12           open.push(s')
13           progress(s')
14         else:
15           analyse(s, t)
16   return False
```

Fig. 7. Reachability analysis

and added to the open list. In addition to the normal verification function, we now call *progress* with the successor state s' (line 13). Pseudo-code for the *progress* function is given in Fig. 8. It determines the reached location of the test automaton which has currently the smallest distance to the test automaton's last location. Remember, if the last location of the test automaton is reachable, then the counterexample is a real counterexample. After the execution of the *verify* function $dead\_end$ is the dead end location.

```
1  function progress(s):
2    if dist(s(test)) < dist(dead_end):
3      dead_end = s(test)
4      u_hint = ∅
```

Fig. 8. On the fly detection of the dead end location

If, during the generation of successor states, a transition t is not enabled it is passed together with its starting state s to the *analyse* function (line 15). Pseudo code for this function is shown in Figure 9. The *analyse* function checks if the test automaton in s is in the current $dead\_end$ location. If this is the case, then it checks if applying t would enable the current dead end transition t_{dead_end}. If this is the case, then *analyse* collects all the variables and clocks respectively that appear in unsatisfied constraints of t's guard. If the set of these variables u is smaller than u_{hint}, then u_{hint} is updated. After the execution of the *verify* function u_{hint} contains variables that hinder the execution of the dead end transition.

After UPPAAL has checked that the counterexample is spurious, all variables that occur in the set of unsatisfied constraints u_{hint} are reported. These variables should not be abstracted in the next iteration, as they hinder the progress of the test automaton. This ensures that the revealed spurious counterexample will not be found in the next iteration. In the following, we explain that the reported variables are likely to be helpful.

```
1  function analyse(s, t):
2    if s(test) ≠ dead_end:
3      return
4    if t_dead_end is synchronised:
5      if t can synchronise with t_dead_end:
6        u = {c ∈ inv(succ(s, t)) | c unsat constraint}∪
7              {c ∈ guard(t) | c unsat constraint}
8        if |u| ≤ |u_hint| ∨ u_hint = ∅:
9          u_hint = u
10   else if assignment of t makes guard(t_dead_end) True:
11     u = {c ∈ inv(succ(s, t)) | c unsat constraint}∪
12           {c ∈ guard(t) | c unsat constraint}
13     if |u| ≤ |u_hint| ∨ u_hint = ∅:
14       u_hint = u
```

Fig. 9. On the fly extraction of least blocking variables. Used expressions: $\mathrm{inv}(s)$: conjunction of s's location invariants, $s(test)$: the location of the test automaton in s, $\mathrm{succ}(s,\, t)$: the successor state of s reached through t, $\mathrm{guard}(t)$: t's guard, $\mathrm{dist}(l)$: distance from a location l of the test automaton to the last location of the test automaton in terms of transitions.

From the construction of the test automaton we know that there are at most two types of transitions in the test automaton: synchronised transitions and τ-transitions. The guard of such a synchronised transition is always satisfiable because it was already satisfied when the starting location of the transition was reached. It only checks that no time elapses since the last transition. Depending on which part the transitions represent from the abstract counterexample, we can distinguish three different cases.

If the dead end transition belongs to one of the transitions introduced for a delay in the abstract counterexample, then the progress of the test automaton is blocked because the full system cannot idle due to an unsatisfied location invariant. This is only possible if this location invariant talks about a clock that was abstracted away because in the abstraction it is possible to take this transition. Therefore this clock should not be abstracted in the next iteration of the abstraction refinement loop.

If the dead end transition belongs to one of the two transitions introduced for a τ-transition in the abstract counterexample, then we know that the progress of the test automaton stops because of a τ-transition in the full system. The reason for this is that the clock guards of the two introduced transitions are satisfied. So the only reason why this may block is that the guard that checks the valuation of the variables is not satisfied. But this is only possible if there is no enabled τ-transition in the full system whose assignments would make the guard true. As there is such a transition in the abstraction, we again know that this must be because of an unsatisfied transition guard. So the variables that occur in the unsatisfied constraints of this guard should not be abstracted in the next iteration.

The last possible reason why the dead end transition is blocked is that there is no enabled transition with an assignment that would make the guard of the dead end transition true. From the construction of the test automaton this can only be a synchronised transition in the full system because a τ-transition in the full system always has to synchronise with the test automaton which is not possible. As we know that there is

a synchronised transition in the abstract system that makes the guard of the dead end transition true, this transition would do it also in the full system. The reason why the progress of the test automaton is stopped is that either the guard of this synchronised transition or a location invariant of the successor state is not satisfied.

5 Experiments

To demonstrate the potential of our approach we chose the "Single-tracked Line Segment" (SLS) case study which stems from the UniForM-project [7]. It is the specification of a control system for a single-tracked line segment for tramways. It is implemented by distributed PLC automata [8]. We took three different models of the SLS case study [8] as examples. As the safety property to verify, we chose the mutual exclusion of drive permissions, i. e., the control system never gives permission to both directions simultaneously.

The first model (M1) we checked is a manipulated system that we obtained by changing a delay time but with the assumption that everything is implemented on only one hardware device. The full UPPAAL model we got from MOBY/RT had 9 processes, 2 clocks and 24 integer variables. Table 1 shows in the first row the resources needed to check the full model of M1 using the standard UPPAAL verification engine. It took 310 seconds to verify that the manipulated delay time does not lead to an error if the system is implemented on one device. In the following rows the steps of the abstraction refinement loop are given. Each step consists of a verification run to find an abstract counterexample (left columns) and the check for spuriousness (right columns). For these runs we use a variant of UPPAAL called UPPAAL/DMC that allows for *directed model checking* [9,10]. Directed model checking is the application of heuristics to model checking and was pioneered a few years ago by Edelkamp et al. [11,12], christening this research direction directed model checking. The use of directed model checking makes sense because it can be expected that the current abstraction contains (abstract) counterexamples and directed model checking detects counterexamples faster. Note that whenever a spurious counterexample is found a refined abstraction is derived. This refined abstraction considers more entities (at least one clock or one integer variable more than before).

For M1 it turns out that with our counterexample guided abstraction refinement we can prove correctness of the model using an abstraction with 1 clock, 4 processes and 14 integer variables less than the full model. The required memory is about 5 % and the *summarized* time consumption is approx. 3 % compared to the full model. Note that in abstraction 3, the number of processes has changed. The reason for this is that whenever a variable is added to the next abstraction that is triggered by the environment, then an additional automaton is added to the system that drives this variable. These driver automata are automatically generated by MOBY/RT.

For the next verification problem we removed the assumption about the partitioning of the PLC automata onto hardware devices. The second experiment (M2) represents a distributed system. Now, the manipulated delay time leads to an incorrect system. However, it was not possible to find a counterexample in the full model within the given memory limit of 2 GBs. This time the abstraction refinement loop had to iterate 8 times to generate an abstraction for which a definite answer was found, i. e., a counterexample

Table 1. Abstraction refinement results for the experiments. Abbreviations: #c: number of clocks, # p: number of processes, # v: number of integer variables, time: runtime in seconds, mem: memory peak in MB, trace: length of found error trace, CE: counterexample.

Model	# c	# p	# v	time	mem	trace	# c	# p	# v	time	mem	result
M1: full							2	9	24	310.0	721	verified
Abstr. #1	0	4	3	0.0	8	20	3	10	24	0.0	7	spurious CE
Abstr. #2	1	4	3	0.0	9	22	3	10	24	3.2	36	spurious CE
Abstr. #3	1	5	5	0.0	9	23	3	10	24	0.4	11	spurious CE
Abstr. #4	1	5	6	0.0	9	23	3	10	24	1.3	20	spurious CE
Abstr. #5	1	5	8	0.0	9	34	3	10	24	2.2	27	spurious CE
Abstr. #6	1	5	10	0.8	9	–						verified
M2: full							3	10	25	> 1527.0	> 2048	out of memory
Abstr. #1	0	5	3	0.0	8	27	4	11	25	0.0	8	spurious CE
Abstr. #2	1	5	3	0.0	9	29	4	11	25	0.0	8	spurious CE
Abstr. #3	2	5	3	0.0	9	30	4	11	25	113.6	491	spurious CE
Abstr. #4	2	5	6	0.1	9	88	4	11	25	44.6	247	spurious CE
Abstr. #5	2	6	8	0.4	9	64	4	11	25	7.6	64	spurious CE
Abstr. #6	2	6	9	0.1	9	44	4	11	25	14.3	108	spurious CE
Abstr. #7	2	6	11	0.2	9	62	4	11	25	15.8	97	spurious CE
Abstr. #8	3	6	11	0.2	9	77	4	11	25	0.5	12	disproved
M3: full							3	10	25	> 1242.0	> 2048	out of memory
Abstr. #1	0	5	3	0.0	8	27	4	11	25	0.0	8	spurious CE
Abstr. #2	1	5	3	0.0	9	29	4	11	25	0.0	8	spurious CE
Abstr. #3	2	5	3	0.0	9	30	4	11	25	117.8	753	spurious CE
Abstr. #4	2	5	6	0.1	9	88	4	11	25	45.3	312	spurious CE
Abstr. #5	2	6	8	0.4	9	64	4	11	25	7.7	80	spurious CE
Abstr. #6	2	6	9	0.1	9	44	4	11	25	14.5	141	spurious CE
Abstr. #7	2	6	11	0.2	9	62	4	11	25	15.9	130	spurious CE
Abstr. #8	3	6	11	3.3	13	–						verified

in the full model. The computed abstraction saved 4 processes and 14 integer variables. The memory and time consumption was *at most* 25 % respectively 14 % compared to the full model.

In our final experiment (M3) we reverted to the original delay time. Again, it was not possible to check the full model within the memory limits. The abstraction refinement loop generated the same sequence of refined abstractions and terminated after 8 iterations again. But this time UPPAAL was able to verify that the final abstraction has no counterexample.

All these experiments show that the abstraction refinement presented in this paper is able to generate abstractions effectively for which definite verification results can be found. The main benefits are that there is no need for human interaction at all, an abstraction of the model is computed automatically for which a reliable verification result can be computed and that this approach reduces significantly the resources (time and memory). However, there is no guarantee that this approach computes a *minimal* abstraction but it is obvious that it will terminate since each iteration adds at least one of the finitely many entities of the model.

6 Related Work

Abstraction refinement was pioneered by Clarke et al. [13] in the early 90s. Since then many researchers have automated this process starting with the work of Balarin and Sangiovanni-Vincentelli [14]. The term counterexample guided abstraction refinement was coined by Clarke et al. [15]. This work deals with discrete timed systems and ACTL* formulas. It has been extended to continuous-time models in [16,17]. The main idea of these approaches is to refine the discrete state space by an appropriate replication of states which have been involved in a spurious counterexample to avoid this trace in the next iteration.

Alur et al. [18] have proposed a predicate abstraction based approach for a counterexample guided abstraction refinement procedure applicable to hybrid systems. The refined abstraction extends the state space by predicates describing additional information on the continuous state space. These additional predicates are constructed by identifying the dead-end state of a spurious trace and an analysis which polyhedron in the continuous state space avoid a reoccurrence of the trace.

A similar approach was proposed by Segelken [19]. Here the analysis of the spurious trace generates an automaton that is put in parallel with the model in the next iteration of the abstraction refinement loop. This automaton represents an infeasible fragment of the previous spurious trace. By the construction of the automaton this infeasible fragment is avoided.

Also in the area of timed automata there are approaches for model checking by iterative refinement of approximations. One of these approaches was implemented in Laroussinie's and Larsen's compositional model checker CMC [20]. This tool starts with a small subset of the automata of the system. It subsequently adds automata to this set and minimises the intermediate result.

Another abstraction refinement approach was developed by Sorea et al. [21,22] in which predicate abstraction was used on the level of the regions which are defined by predicates over clocks. The approach uses symbolic counterexamples from failed model-checking attempts. Such a symbolic counterexample represents a sequence of sets of states, and can be seen as generalisation of a linear counterexample. To exclude a spurious symbolic counterexamples from further iterations, new abstraction predicates are chosen randomly from a set of predefined predicates. Except for the fact that new abstraction predicates are chosen randomly, this approach is, in some respect, quite similar to what we are proposing here. The main differences are the nature of the counterexamples and that new abstraction variables are selected more carefully. Unfortunately the authors do not give any runtime results.

7 Conclusion and Future Work

We presented an approach for counterexample guided abstraction refinement for a subclass of timed automata. In this paper we defined how to construct test automata that can be used to check whether a full model is able to behave as the abstract trace, i. e., to check whether an abstract trace is spurious or not. Moreover, we extended the model checker UPPAAL in such a way that it executes an analysis of *why* a full model cannot

execute a spurious trace. The result of this analysis is used to refine the given abstraction in a way that the spurious counterexample cannot occur anymore. This approach enabled us to construct a closed abstraction refinement loop in which verification of a system starts with the coarsest abstraction and with each iteration the abstraction is refined until a result is found that holds for the full model, too.

In its current version our approach is able to refine the abstraction by adding variables or clocks. From our point of view, the most promising direction of future work is to extend the approach such that it also refines the set of automata considered in the abstraction. At present the set of PLC automata is fixed. However, if a system consists of many parallel components it makes sense to start with a small subset thereof. Since the semantics of a subset is an over-approximation this fits our approach. Then the abstraction refinement analysis needs to be extended accordingly in a way that it can also identify PLC automata for the abstraction refinement. Having such an extension the abstraction refinement loop would start with the coarsest abstraction of only those automata that manipulate variables appearing in the requirement.

In this paper we have presented and implemented an abstraction refinement methodology for PLC automata. However, the methodology is generally applicable to the full range of timed automata based models expressible within UPPAAL. Here, a particularly challenge will be the generalisation of the automated analysis of counterexamples presented in Section 4 to deal with the rich imperative language (including structured data types, user-defined types as well as user-defined functions and procedures) provided in UPPAAL 4.0. We envisage the need for incorporating the work of Sørensen and Trane on slicing UPPAAL 4.0 models [23].

References

1. Dierks, H.: Time, Abstraction and Heuristics – Automatic Verification and Planning of Timed Systems using Abstraction and Heuristics. Technical report, University of Oldenburg (2006)
2. Olderog, E.R., Dierks, H.: Moby/RT: A Tool for Specification and Verification of Real-Time Systems. J. UCS 9, 88–105 (2003)
3. Dierks, H.: Specification and Verification of Polling Real-Time Systems. PhD thesis, University of Oldenburg (1999)
4. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126, 183–235 (1994)
5. Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: Paterson, M.S. (ed.) Automata, Languages and Programming. LNCS, vol. 443, pp. 322–335. Springer, Heidelberg (1990)
6. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
7. Krieg-Brückner, B., Peleska, J., Olderog, E.R., Baer, A.: The uniform workbench, a universal development environment for formal methods. In: Woodcock, J.C.P., Davies, J., Wing, J.M. (eds.) FM 1999. LNCS, vol. 1709, pp. 1186–1205. Springer, Heidelberg (1999)
8. Dierks, H.: PLC-Automata: A New Class of Implementable Real-Time Automata. Theor. Comput. Sci. 253, 61–93 (2001)
9. Kupferschmid, S., Hoffmann, J., Dierks, H., Behrmann, G.: Adapting an AI planning heuristic for directed model checking. In: Valmari, A. (ed.) Model Checking Software. LNCS, vol. 3925, pp. 35–52. Springer, Heidelberg (2006)

10. Kupferschmid, S., Dräger, K., Hoffmann, J., Finkbeiner, B., Dierks, H., Podelski, A., Behrmann, G.: UPPAAL/DMC – Abstraction-based Heuristics for Directed Model Checking. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 679–682. Springer, Heidelberg (2007)
11. Edelkamp, S., Lluch-Lafuente, A., Leue, S.: Directed explicit-state model checking in the validation of communication protocols. STTT (2004)
12. Edelkamp, S., Lluch-Lafuente, A., Leue, S.: Directed explicit model checking with HSF-SPIN. In: Dwyer, M.B. (ed.) Model Checking Software. LNCS, vol. 2057, pp. 57–79. Springer, Heidelberg (2001)
13. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. 16, 1512–1542 (1994)
14. Balarin, F., Sangiovanni-Vincentelli, A.L.: An iterative approach to language containment. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 29–40. Springer, Heidelberg (1993)
15. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
16. 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. Found. Comput. Sci. 14, 583–604 (2003)
17. Fehnker, A., Clarke, E., Jha, S., Krogh, B.: Refining Abstractions of Hybrid Systems using Counterexample Fragments. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, Springer, Heidelberg (2005)
18. Alur, R., Dang, T., Ivancic, F.: Predicate Abstraction for Reachability Analysis of Hybrid Systems. Trans. on Embedded Computing Sys. 5, 152–199 (2006)
19. Segelken, M.: Abstraction and Counterexample-guided Construction of Omega-Automata for Model Checking of Step-discrete linear Hybrid Models. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, Springer, Heidelberg (2007)
20. Laroussinie, F., Larsen, K.G.: CMC: A tool for compositional model-checking of real-time systems. In: Proc. FORTE/PSTV, pp. 439–456. Kluwer Academic Publishers, Dordrecht (1998)
21. Möller, M.O., Rueß, H., Sorea, M.: Predicate abstraction for dense real-time system. In: Proc. TPTS, Elsevier, Amsterdam (2002)
22. Sorea, M.: Lazy approximation for dense real-time systems. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 363–378. Springer, Heidelberg (2004)
23. Sørensen, U., Trane, C.: Optimization for the Uppaal verification tool. Technical report, Aalborg University (2007)

Dynamical Properties of Timed Automata Revisited

Cătălin Dima

LACL, Université Paris 12, 61 av. du Général de Gaulle,
94010 Créteil, France

Abstract. We give a generalization of a solution by Puri to the problem of checking emptiness in timed automata with drifting clocks for the case of automata with non-closed guards. We show that non-closed guards pose certain specific problems which cannot be handled by Puri's algorithm, and propose a new algorithm, based on the idea of "boundary clock regions" of Alur, LaTorre and Pappas. We then give a symbolic algorithm for solving the reachability problem. Our algorithm is based on a symbolic construction of the "neighborhood" of a zone, and on a procedure that, given a set of zones $\mathcal{Z}$, builds the forward propagation of the strongly connected components which can be reached from $\mathcal{Z}$. This improves a symbolic algorithm of Daws and Kordy, due to the ability to handle sets of zones.

1 Introduction

Timed automata [AD94] are a widely accepted and powerful model of real time systems. They are finite automata endowed with dense-time variables, called *clocks*, that are used to measure time intervals separating actions. Dense time is utilized as an abstraction, in the sense that the model is not sensitive w.r.t. the "clock tick" in the implementation. Efficient algorithms and tools [LPY97, BDM+98, HHWT97] have been designed and applied with succes for the verification of safety properties of systems modeled as timed automata.

Clocks in timed automata are synchronous: letting time pass t units increases *all* clocks with t. This assumption is sometimes too strong in distributed systems, in which some degree of non-synchronicity between the local clocks of each component may be present. The rectangular automata of [HKPV98] utilize dense variables that increase at a rate in some interval $[\alpha, \beta]$, and the subclass of initialized rectangular automata has a decidable reachability problem. Initialized rectangular automata may give a model of "inexact" timed automata with known clock drift Δ, if the rate binding interval is always $[1-\Delta, 1+\Delta]$. Hence, timed automata with a *known* clock drift have a decidable reachability problem.

In this paper, we are interested in solving the following *safe implementation problem:* given a timed automaton $\mathcal{A}$ and some zone Z, does there exist Δ for which no trajectory in which clocks drift with at most Δ units reaches Z? In [Pur98], A. Puri has shown how to compute the set of states that are reachable for any clock drift Δ. [Pur98] showed that, in the case of closed constraints, the region automaton does not provide the complete answer, as one needs to consider cycles in the (closed) region graph that have a nonempty intersection with regions

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 130–146, 2007.

that are already reachable. This result was further extended, in [DK06], where a symbolic algorithm for constructing a "stable zone" in the region graph is given. The stable zone is constructed for each cycle in the automaton graph that can be reached. Both the work in [Pur98] and in [DK06] treat of automata with closed clock constraints.

In this paper, we extend these results in two directions. First, we investigate the extension of the technique of Puri to automata with non-closed constraints. Secondly, we give a symbolic algorithm that constructs "sets of stable zones" as (forward propagations of) sets of zones which lie on some cycle in the region graph. Our technique can be applied to data structures for representing *sets of zones*, like in [LPWY99].

Our extension of Puri's technique to non-closed guards utilizes a variant of *boundary clock regions* of [ATP01], to model the fact that trajectories that pass through some region R can be arbitrarily close to some region R' neighboring R. Note that, as in [Pur98], we work with automata with bounded constraints; another assumption is that discrete transitions in a run are separated by non-zero delays.

Recently, [DDR05b, AT05] addressed a similar class of problems: given a timed automaton $\mathcal{A}$, does there exist some $\Delta > 0$ and a Δ-drift clock implementation of $\mathcal{A}$, in which "important" properties of $\mathcal{A}$ be preserved? [DDR05b, DDR05a] consider this problem in the context of verifying whether a controller C specified as a timed-automaton can be implemented by some drifting-clock automaton. Their approach is to model the system composed of the controller and the environment it must control as a parametric rectangular automaton in which Δ is a parameter. The drawback of this approach is that parametric model checking of timed automata with three clocks and only one parameter is known to be undecidable [AHV93, WT97]. Hence, tools like Uppaal cannot be directly applied to synthesize the value of Δ. The approach proposed in [DDR05a] is to guess an initial value for Δ and check it with Uppaal; if this guess satisfies the desired properties, then, according to the results of [DDR05b], any "faster" implementation (with $\Delta' < \Delta$) is also correct. This guess could be avoided by using the techniques from [DK06] and this paper.

The rest of the paper is divided as follows: in the next section we recall the definition and basic facts about timed automata and their drifting semantics. Section 3 contains the construction of the *boundary region automaton* and its correctness. Section 4 is devoted to the presentation of the symbolic algorithm and to comments on the improvements of our approach w.r.t. [DK06]. We end with a section with conclusions.

2 Timed Automata

A **timed automaton.** [AD94] is a tuple $\mathcal{A} = (Q, \mathcal{X}, \delta, Q_0, Q_f)$ where Q is a finite set of *locations*, $\mathcal{X}$ is a finite set of *clocks*, $Q_0, Q_f \subseteq Q$ are sets of *initial*, resp. *final* locations, and δ is a finite set of tuples called *transitions*, (q, C, X, q'), where $q, q' \in Q$, $X \subseteq \mathcal{X}$, and C is a finite conjunction of *simple constraints* utilizing

clocks as variables – that is, constraints of the form $x \in I$, where I is an interval with nonnegative integer bounds. We will consider in this paper only *bounded* intervals, i.e. *excluding* intervals of the form $[2, \infty[$. For each $(q, C, X, r) \in \delta$, the component C is called the *guard* of the transition and X is its *reset component*. We consider the set of clocks is ordered as $\mathcal{X} = \{x_1, \ldots, x_n\}$.

In the standard semantics $\mathcal{A}$ can make time-passage transitions, in which all clocks advance with the same amount of time, and discrete transitions, in which location may change. The last are enabled when the "current clock valuation" satisfies the guard C of a transition (q, C, X, q'), and when they are executed, the clocks in the "reset component" X are set to zero. The notations used henceforth are the following: for a given point $v \in \mathbb{R}^n_{\geq 0}$ and $X \subseteq \mathcal{X}$, $v[X := 0]$ is the point obtained by reseting all clocks in X, defined by $(v[X := 0])_i = v_i$ for $x_i \notin X$ and $(v[X := 0])_i = 0$ for $x_i \in X$. We will also denote $\mathbf{0}_n$ the origin point, i.e. $(\mathbf{0}_n)_i = 0$ for all $1 \leq i \leq n$.

In the drifting semantics [Pur98, DDR05b], when time advances by t, each clock advances with some $t' \in [t(1-\Delta), t(1+\Delta)]$, independently of the others, $\Delta > 0$ denoting the maximal clock drift. The **Δ-drifting semantics** of $\mathcal{A}$ is the timed transition system $\mathcal{T}_\Delta(\mathcal{A}) = (\mathcal{Q}, \theta_\Delta, \mathcal{Q}_0, \mathcal{Q}_f)$ where $\mathcal{Q} = Q \times \mathbb{R}^n_{\geq 0}$, $\mathcal{Q}_0 = Q_0 \times \{\mathbf{0}_n\}$ (all clocks are set to *zero initially*), $\mathcal{Q}_f = Q_f \times \mathbb{R}^n_{\geq 0}$ and

$$\theta_\Delta = \big\{(q, v) \xrightarrow{t}_\Delta (q, v') \mid t > 0, v'_i - v_i \in \big[t(1-\Delta), t(1+\Delta)\big]\ \forall 1 \leq i \leq n\big\}$$
$$\cup \big\{(q, v) \xrightarrow{\downarrow}_\Delta (q', v[X := 0]) \mid \exists (q, C, X, q') \in \delta \text{ such that } v \models C\big\}$$

Here $\models$ denotes the usual satisfiaction relation for clock constraints. Elements of $\mathcal{Q}$ are called **states**. When the automaton $\mathcal{A}$ is fixed, we use $\mathcal{T}_\Delta$ for $\mathcal{T}_\Delta(\mathcal{A})$.

A **$\mathcal{T}_\Delta$-trajectory** is a sequence of transitions $\tau = \big((q_{i-1}, v_{i-1}) \xrightarrow{\xi_i}_\Delta (q_i, v_i)\big)_{1 \leq i \leq k}$ in θ_Δ, with $\xi_i \in \mathbb{R}_{>0} \cup \{\downarrow\}$. We denote this situation as $(q_0, v_0) \overset{\tau}{\rightsquigarrow}_{\mathcal{T}_\Delta} (q_k, v_k)$. Also, we denote $(q, v) \rightsquigarrow_{\mathcal{T}_\Delta} (q', v')$ when there exists a $\mathcal{T}_\Delta$-trajectory τ such that $(q, v) \overset{\tau}{\rightsquigarrow}_{\mathcal{T}_\Delta} (q', v')$. Trajectory τ is **accepting** if it starts in $\mathcal{Q}_0$ and ends in $\mathcal{Q}_f$. The set of $\mathcal{T}_\Delta$-trajectories is denoted Traj_Δ.

A **run** in $\mathcal{A}$ is a sequence $\rho = \big((q_{i-1}, C_i, X_i, q_i)\big)_{1 \leq i \leq k}$ of transitions from δ. A run $\rho = \big((q_{i-1}, C_i, X_i, q_i)\big)_{1 \leq i \leq k}$ is **associated with** a $\mathcal{T}_\Delta$-trajectory $\tau = \big((\overline{q}_{i-1}, v_{i-1}) \xrightarrow{\xi_i} (\overline{q}_i, v_i)\big)_{1 \leq i \leq l}$ if $l = 2k$ or $l = 2k+1$ and for each $1 \leq i \leq k$, $\overline{q}_{2i} = \overline{q}_{2i+1} = q_i$, $\xi_{2i-1} \in \mathbb{R}_{>0}$, $\xi_{2i} = \downarrow$, $v_{2i} = v_{2i-1}[X_i := 0]$, $v_{2i-1} \models C_i$, and also $\xi_{2k+1} \in \mathbb{R}_{>0}$, $\overline{q}_0 = \overline{q}_1 = q_0$.

For each $\Delta > 0$ and state $(q, v) \in \mathcal{Q}$, the set of **$\mathcal{T}_\Delta$-reachable states** from (q, v) is:

$$\mathsf{Reach}_\Delta(q, v) = \big\{(q', v') \in \mathcal{Q} \mid (q, v) \rightsquigarrow_{\mathcal{T}_\Delta} (q', v')\big\}$$

The **reachable states in the limit** from (q, v) are:

$$\mathsf{Reach}_{\Delta \to 0}(q, v) = \bigcap\nolimits_{\Delta > 0} \mathsf{Reach}_\Delta(q, v)$$

By extension, for any $S \subseteq \mathbb{R}^n_{\geq 0}$, we denote $\mathsf{Reach}_\Delta(q,S) = \bigcup_{v\in S} \mathsf{Reach}_\Delta(q,v)$ and $\mathsf{Reach}_{\Delta\to 0}(q,S) = \bigcup_{v\in S} \mathsf{Reach}_{\Delta\to 0}(q,v)$.

Throughout this paper we assume that there exists a clock x which is reset at each transition and which is checked, on each transition, to be greater than zero. Note that this assumption implies the fact that each cycle in the timed automaton contains a clock reset, as in [Pur98]. We also consider that $\mathcal{A}$ has no self loops. Note also that the semantics of $\mathcal{T}_\Delta$ (in which time steps have non-zero duration) also implies that time must strictly progress within each cycle, as required in [DDMR04]. It is well-known that any timed automaton can be transformed syntactically into an automaton satisfying these assumptions.

Regions and region reachability. A **zone** [Yov98] is an n-dimensional convex set of points which can be uniquely represented by a diagonal constraint of the form $C_Z = \bigwedge_{0\leq i,j\leq n}(x_i - x_j \in I_{ij})$, where $x_0 = 0$ and I_{ij} are intervals with integer bounds satisfying the following *triangle inequality*: $\forall 1 \leq i,j,k \leq n, I_{ik} \subseteq I_{ij} + I_{jk}$. The constraint C_Z is called the **normal form representation** of Z.

For $M \in \mathbb{N}$, an M-**region** (or simply a **region**, when M is understood from the context) is a zone R for which the intervals in the normal form representation C_R are either point intervals $I_{ij} = \{a\}$ with $-M \leq a \leq M$, or open unit intervals $I_{ij} =]a, a+1[$ with $-M \leq a \leq M-1$ ($a \in \mathbb{N}$).

Remark 1. Throughout this paper $M_\mathcal{A}$ will denote the largest constant occurring in a constraint in $\mathcal{A}$. We denote $\mathsf{Reg}_\mathcal{A}$ the set of $M_\mathcal{A}$-regions.

The **region automaton** is then $\mathcal{R}_\mathcal{A} = \big(Q \times \mathsf{Reg}_\mathcal{A}, \delta_\mathcal{R}, \mathcal{R}_0, \mathcal{R}_f\big)$ where $\mathcal{R}_0 = \big\{(q, \mathbf{0}_n) \mid q \in Q_0\big\}$, $\mathcal{R}_f = \big\{(q,R) \mid q \in Q_F\big\}$ and

$$\delta_\mathcal{R} = \Big\{(q,R) \xrightarrow{\mathtt{t}} (q,R') \mid R \neq R', \exists v \in R, v' \in R', t \in \mathbb{R}_{>0} \text{ s.t. } (q,v) \xrightarrow{t} (q,v') \text{ and } \forall 0 < t' < t, \forall v'' \in \mathbb{R}^n_{\geq 0} \text{ if } (q,v) \xrightarrow{t'} (q,v''), \text{ then } v'' \in R \cup R'\Big\}$$
$$\cup \Big\{(q,R) \xrightarrow{\downarrow} (q',R') \mid \exists v \in R, v' \in R', \text{ s.t. } (q,v) \xrightarrow{\downarrow} (q',v')\Big\}$$

$\xrightarrow{\mathtt{t}}$ denotes here the *immediate time successor relation*, the time successor relation from [AD94] is denoted $\xrightarrow{\mathtt{t}^*}$. A *run* in $\mathcal{R}_\mathcal{A}$ is a sequence of transitions from $\delta_\mathcal{R}$. Tuples $(q,R) \in Q \times \mathsf{Reg}_\mathcal{A}$ will be called *state regions*.

It is well-known [AD94] that the *region automaton* is a faithful representation of the set of reachable states of $T_0(\mathcal{A})$: there exists a reachable final state $(q,v) \in \mathcal{Q}_f$ iff there exists a reachable state region (q,R) in $Q \times \mathsf{Reg}_\mathcal{A}$ with $v \in R$.

Figure 1 gives an example of a timed automaton and its associated region automaton. The dashed line gives the only transition between a state region of the form (q_0,R) to a state region of the form (q_1,R). Note that no final region is reachable from $(q_0,\mathbf{0}_2)$ in this automaton.

For each state region R and location $q \in Q$ we denote $\mathsf{RegReach}_\Delta(q,R)$ the set of state regions (q',R') that can be touched by a trajectory of $\mathcal{T}_\Delta$ that starts in (q,R), with $\Delta > 0$ fixed; we also denote $\mathsf{RegReach}_{\Delta\to 0}(q,R)$ the set of state regions (q',R') for which, *for each* $\Delta > 0$, there exists a trajectory in $\mathcal{T}_\Delta$ starting

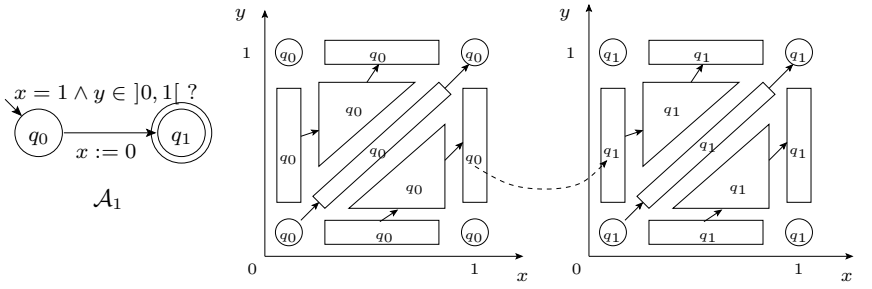


Fig. 1. The timed automaton $\mathcal{A}_1$ and its associated region automaton

in (q, R) that touches (q', R'); these notations are also extended to zones Z:

$$\mathsf{RegReach}_{\Delta}(q, R) = \big\{(q', R') \mid \exists (q', v) \in \mathsf{Reach}_{\Delta}(q, R), v \in R'\big\}$$
$$\mathsf{RegReach}_{\Delta\to 0}(q, R) = \bigcap\nolimits_{\Delta>0} \mathsf{RegReach}_{\Delta}(q, R)$$
$$\mathsf{RegReach}_{\Delta}(q, Z) = \bigcup \big\{\mathsf{RegReach}_{\Delta}(q, R) \mid R \in \mathsf{Reg}_{\mathcal{A}}, R \subseteq Z\big\}$$
$$\mathsf{RegReach}_{\Delta\to 0}(q, Z) = \bigcup \big\{\mathsf{RegReach}_{\Delta\to 0}(q, R) \mid R \in \mathsf{Reg}_{\mathcal{A}}, R \subseteq Z\big\}$$

Example 1. Consider again the timed automaton in Figure 1 and the region R_1 defined by the constraint $C_{R_1} = 0 < x < y < 1$. Then, for any $\Delta > 0$, $\big\{(q_1, v_2) \mid (q_0, \mathbf{0}_2) \rightsquigarrow_{\Delta} (q_1, v_2)\big\} \cap \big(\{q_1\} \times R_1\big) \neq \emptyset$, and therefore $(q_1, R_1) \in \mathsf{RegReach}_{\Delta}(q_0, \mathbf{0}_2)$. This implies that $(q_1, R_1) \in \mathsf{RegReach}_{\Delta\to 0}(q_0, \mathbf{0}_2)$.

Note also that, in the region automaton for $\mathcal{A}_1$, the state region (q_1, R_1) is unreachable from $(q_0, \mathbf{0}_2)$, hence $\mathsf{RegReach}_{\Delta\to 0}(q_0, \mathbf{0}_2) \supsetneq \mathsf{RegReach}_{0}(q_0, \mathbf{0}_2)$.

Consider now the following *safe implementation problem*:

Problem 1. Given a zone Z and a location $q \in Q$, does there exist a clock drift Δ for which no trajectory in $\mathcal{T}_{\Delta}$ reaches a state (q, v) with $v \in Z$?

Note that the safe implementation problem is not equivalent with checking whether $\mathsf{Reach}_{\Delta\to 0}(q_0, \mathbf{0}_n) \cap (q, Z) \neq \emptyset$: in $\mathcal{A}_1$ from Figure 1, for any $\Delta > 0$, if $(q_0, \mathbf{0}_2) \xrightarrow{t}_{\Delta} (q_0, v_1) \xrightarrow{\downarrow}_{\Delta} (q_1, v_2)$ for some clock valuations $v_1, v_2 \in [0,1]^2$ then $v_1(y) = v_2(y) \in [1-\Delta, 1[$ and, therefore, for the region R_2 defined by $C_{R_2} = (x = 0 \wedge y = 1)$,

$$\mathsf{Reach}_{\Delta\to 0}(q_0, \mathbf{0}_2) \cap (q_1, R_2) = \bigcap\nolimits_{\Delta>0} \big\{(q_1, v_2) \mid (q_0, \mathbf{0}_2) \rightsquigarrow_{\Delta} (q_1, v_2), v_2 \in R_2\big\}$$
$$\subseteq \bigcap\nolimits_{\Delta>0} \{q_1\} \times \big(([0,1] \times [1-\Delta, 1[\,) \cap R_2\big) = \emptyset$$

On the other hand, Example 1 above shows that $(q_1, R_1) \in \mathsf{RegReach}_{\Delta\to 0}(q_0, \mathbf{0}_2)$ and hence $\mathsf{RegReach}_{\Delta\to 0}(q_0, \mathbf{0}_2) \cap (q_1, R_2) \neq \emptyset$.

More interestingly, we may also observe that

$$\mathsf{Reach}_{\Delta\to 0}(q_0, \mathbf{0}_2) \cap (q_1, [0,1]^2)$$
$$\subseteq \bigcap\nolimits_{\Delta>0} \{q_1\} \times \Big(\big(\{0\} \times [1-\Delta, 1[\,\big) \cup \big(]0, \frac{\Delta+\Delta^2}{1-\Delta}] \times [1-\Delta, 1]\big)\Big) = \emptyset$$

which actually means that no state in the state region (q_1, R) can be Δ-reached for any Δ. Hence the pure study of $\mathsf{Reach}_{\Delta\to 0}$ is insufficient for solving the safe implementation problem.

The example with region R_2 also suggests that "closing the guards" in the given timed automata would not work. By closing the guards, we mean here the transformation of each automaton $\mathcal{A}$ into a timed automaton $\overline{\mathcal{A}}$ in which each transition copies a transition of $\mathcal{A}$, but with all constraints transformed into non-strict. To see that this technique does not work in general, note that, in $\overline{\mathcal{A}}_1$, $(q_1, R_2) \in \mathsf{Reach}_0(q_0, \mathbf{0}_n) \setminus \mathsf{RegReach}_{\Delta\to 0}(q_0, \mathbf{0}_n)$.

Before ending this section, we give a useful technical property relating convexity with runs in the region automaton. This property utilizes the following notion of *association* between runs in the timed automaton and runs in the region automaton: a run $\rho = ((q_{i-1}, R_{i-1}) \xrightarrow{\xi_i} (q_i, R_i))_{1\le i\le k}$ in $\mathcal{R}_{\mathcal{A}}$ is *associated* with a run $\rho' = \big((r_{j-1}, C_j, X_j, r_j)\big)_{1\le i\le m}$ in $\mathcal{A}$ if there are m indices $j_1, \ldots, j_m \le k$ such that $\xi_{j_l} = \downarrow$, $R_{j_l} = R_{j_{l-1}}[X := 0]$ and $R_{j_{l-1}} \subseteq Z_{C_l}$ $(1 \le l \le m)$, and also $\xi_i = \mathtt{t}$ for all $i \neq j_1, \ldots, j_m$. In other words, ρ and ρ' are associated iff all trajectories subsumed by ρ are associated with ρ'.

For each bounded region $R \in \mathsf{Reg}_{\mathcal{A}}$, we denote by $V(R)$ the set of vertices, or *cornerpoints* that bound R. For example, for the 2-dimensional region $0 < x < y < 1$, $V(R) = \{(0,0), (0,1), (1,1)\}$. Note that $V(R)$ can be formally defined using the "fractional part" [AD94] representation of regions.

Proposition 1. *Suppose ρ_1 and ρ_2 are two runs in $\mathcal{R}_{\mathcal{A}}$, with ρ_i starting in (q, R_i') and ending in (q', R_i'') $(i = 1, 2)$. Suppose that both runs are associated with the same run ρ in $\mathcal{A}$ and that there exist R, R' such that $V(R) = V(R_1) \cup V(R_2)$, $V(R') = V(R_1') \cup V(R_2')$. Then there is a run ρ' in $\mathcal{R}_{\mathcal{A}}$ that starts in (q, R_1), ends in (q', R_2) and is associated with ρ.*

The proof of this property is based on zone convexity.

3 The Extended Boundary Region Automaton

First, let us recall here briefly Puri's technique for constructing the set of reachable regions in a closed timed automaton: the $\Delta \to 0$-reachable regions are obtained by applying, alternatively, the following two procedures until a fixpoint is reached:

1. Forward closure of a given set of regions.
2. Add cycles in the "closed region automaton", that have a nonempty intersection with an already reachable region,

In the previous section, we have seen that this technique cannot be applied as is to the closure $\overline{\mathcal{A}}$ of the given automaton $\mathcal{A}$, due to inherent peculiarities of working with non-closed guards.

In this section, we refine Puri's technique of searching for cycles in the region graph, by carefully defining when to consider that a (possibly open) region

"touches" a reachable region. The right notion of "touching" is given in the following definition:

Definition 1. *A region R is* **t-*aligned*** *if its normal form representation $C_R = \bigwedge_{0\le i,j\le n} x_i - x_j \in I_{ij}$ has the property that I_{i0} is not a point interval for all i. Equivalently, for any $q \in Q$, there exist $v, v' \in R$ with $(q,v) \xrightarrow{t}_0 (q,v')$ for some $t > 0$.*

Two regions R, R' are ***neighbors*** *if $V(R) \cap V(R') \neq \emptyset$. R, R' are* **t-*neighbors*** *if both are* t*-aligned and either $V(R') \subseteq V(R)$ or $V(R) \subseteq V(R')$.*

Example 2. For any timed automaton, the region R_1 with $C_{R_1} : 0<x<y<1$ is a t-neighbor for R_2, with $C_{R_2} : 0<x=y<1$, while $\mathbf{0}_2$ is not a t-neighbor of R_2.

The idea behind the computation of $\mathsf{RegReach}_{\Delta\to 0}$ is to utilize, instead of regions, pairs of regions (R, R') with $V(R) \supseteq V(R')$. Such pairs are similar to the *boundary regions* of [ATP01]: they model sets of trajectories that pass through R and are "arbitrarily close" to R'. Formally we construct the *boundary regions automaton* $\mathcal{E}(\mathcal{A}) = \{\mathcal{Q}_{\mathcal{E}(\mathcal{A})}, \theta_{\mathcal{E}(\mathcal{A})}, \mathcal{Q}^b_0, \mathcal{Q}^b_f\}$ where $\theta_{\mathcal{E}(\mathcal{A})} \subseteq \mathcal{Q}_{\mathcal{E}(\mathcal{A})} \times \mathcal{Q}_{\mathcal{E}(\mathcal{A})}$ and

$$
\begin{aligned}
\mathcal{Q}_{\mathcal{E}(\mathcal{A})} &= \{(q,R,R') \mid q \in Q, R, R' \in \mathsf{Reg}, V(R) \supseteq V(R')\}\\
\mathcal{Q}^b_0 &= \{(q,\mathbf{0}_n,\mathbf{0}_n) \in \mathcal{Q}_{\mathcal{E}(\mathcal{A})} \mid q \in Q_0\} \quad \text{and} \quad \mathcal{Q}^b_f = \{(q,R,R') \in \mathcal{Q}_{\mathcal{E}(\mathcal{A})} \mid q \in Q_f\}\\
\theta_{\mathcal{E}(\mathcal{A})} &= \{(q,R_1,R) \xrightarrow{n}_{\mathcal{E}(\mathcal{A})} (q,R_2,R) \mid R_1, R_2 \text{ are t-neighbors}\}\\
&\cup \{(q,R_1,R) \xrightarrow{\mathtt{t}_1}_{\mathcal{E}(\mathcal{A})} (q,R_2,R) \mid R_1 \xrightarrow{\mathtt{t}} R_2\}\\
&\cup \{(q,R,R_1) \xrightarrow{r}_{\mathcal{E}(\mathcal{A})} (q,R,R_2) \mid V(R_1) \supseteq V(R_2)\}\\
&\cup \{(q,R,R_1) \xrightarrow{\mathtt{t}_2}_{\mathcal{E}(\mathcal{A})} (q,R,R_2) \mid R_1 \xrightarrow{\mathtt{t}} R_2\}\\
&\cup \{(q_1,R_1,R'_1) \xrightarrow{\downarrow}_{\mathcal{E}(\mathcal{A})} (q_2,R_2,R'_2) \mid (q_1,R_1) \xrightarrow{\downarrow} (q_2,R_2) \in \delta_{\mathcal{R}} \text{ and}\\
&\qquad \text{if } R_2 = R_1[X := 0] \text{ for some } X \subseteq \mathcal{X}, \text{ then } R'_2 = R'_1[X := 0]\}\\
&\cup \{(q,R_1,R'_1) \xrightarrow{\circlearrowleft}_{\mathcal{E}(\mathcal{A})} (q,R_2,R'_2) \mid R'_1, R'_2 \text{ are t-neighbors}, ((q,R'_2),(q,R'_2)) \in \delta^+_{\mathcal{R}}\}
\end{aligned}
$$

($\delta^+_{\mathcal{R}}$ is the transitive closure of the transition relation in the region automaton.)

Elements of $\mathcal{Q}_{\mathcal{E}(\mathcal{A})}$ will be called *boundary regions*. Each type of transition in $\mathcal{E}(\mathcal{A})$ is labeled differently, due to its particular significance: transitions $\xrightarrow{n}$ are between t-**n**eighboring boundary regions, $\xrightarrow{\mathtt{t}_1}$ and $\xrightarrow{\mathtt{t}_2}$ are the two types of t**i**me-passage transitions, while $\xrightarrow{r}$ represent **r**eductions to a smaller boundary region. The reflexive-transitive closure of $\theta_{\mathcal{E}(\mathcal{A})}$ will be denoted $\to_{\mathcal{E}(\mathcal{A})}$, while subsets of it involving only certain types of transitions are identified by their respective symbols. For example, $\xrightarrow{\mathtt{t}_1,\mathtt{t}_2,n}_{\mathcal{E}(\mathcal{A})} = \left(\xrightarrow{\mathtt{t}_1}_{\mathcal{E}(\mathcal{A})} \cup \xrightarrow{\mathtt{t}_2}_{\mathcal{E}(\mathcal{A})} \cup \xrightarrow{n}_{\mathcal{E}(\mathcal{A})}\right)^*$.

We will say that the transition $(q_1,R_1,R'_1) \xrightarrow{\downarrow}_{\mathcal{E}(\mathcal{A})} (q_2,R_2,R'_2)$ is *associated with* a transition $\tau = (q_1,C,X,q_2) \in \delta$ if $R_1 \subseteq Z_C$ and $R_2 = R_1[X := 0]$.

An example is provided in Figure 2 for $\mathcal{E}(\mathcal{A}_1)$, where $\mathcal{A}_1$ is the automaton from Figure 1. (Some of the transitions are labeled only with one of the types they may carry.) Note that, starting from $(q_0,\mathbf{0}_2,\mathbf{0}_2)$, the only reachable boundary

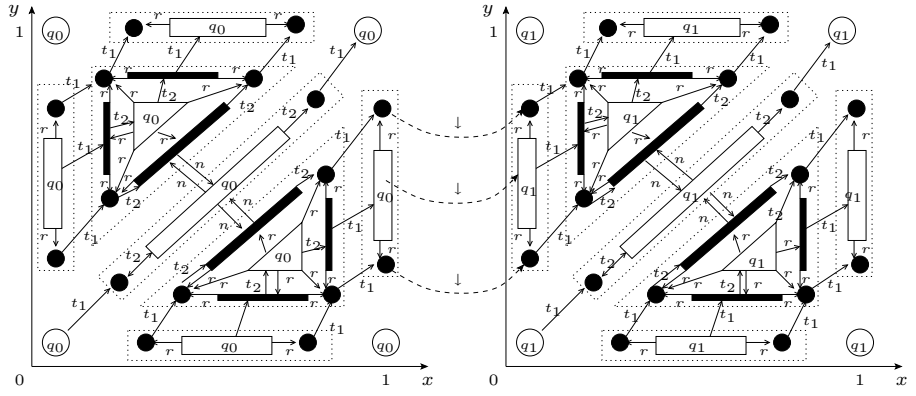


Fig. 2. $\mathcal{E}(\mathcal{A}_1)$ for the automaton in Figure 1

regions in $\mathcal{E}(\mathcal{A}_1)$ in which location q_1 occurs are of the type (q_1, R, R') in which $C_{R'} : (x = 0 \wedge y = 1)$ and $R \neq R'$. As we will see, this is consistent with the fact that $\mathsf{RegReach}_{\Delta\to 0}(q_0, \mathbf{0}_2) \not\ni (q_1, R'')$ where $C_{R''} : (x = 0 \wedge y = 1)$.

The following property says that the third components in boundary regions always follow the transitions of the closure $\overline{\mathcal{A}}$ of the given timed automaton $\mathcal{A}$:

Lemma 1. *If* $(q_1,R_1,R'_1) \xrightarrow{n,\mathtt{t}_1,\mathtt{t}_2,r,\downarrow}_{\mathcal{E}(\mathcal{A})} (q_2,R_2,R'_2)$ *then* $((q_1,R'_1),(q_2,R'_2)) \in \delta^*_{\overline{\mathcal{A}}}$.

The proof is by straightforward induction on the length of the $\mathcal{E}(\mathcal{A})$-run.

Denote $\mathsf{Reach}_{\mathcal{E}(\mathcal{A})}(q_0, \mathbf{0}_n, \mathbf{0}_n)$ the set of boundary regions that can be reached from $(q_0, \mathbf{0}_n, \mathbf{0}_n)$ in $\mathcal{E}(\mathcal{A})$. The first main result of this paper is the following:

Theorem 1. *Let $\mathcal{A}$ be a bounded timed automaton with no self loops and in which there exists a clock x such that all transitions (q, C, X, q') have $x \in X$ and $C \wedge (x = 0)$ not satisfiable. Then:*

$$\mathsf{RegReach}_{\Delta\to 0}(q_0, \mathbf{0}_n) = \big\{(q, R) \mid \exists R' \in \mathsf{Reg}_{\mathcal{A}}, (q, R, R') \in \mathsf{Reach}_{\mathcal{E}(\mathcal{A})}(q_0, \mathbf{0}_n, \mathbf{0}_n)\big\}$$

The inverse inclusion is a corollary of the following technical property:

Proposition 2. *Suppose* $(q_1, R_1, R'_1) \to_{\mathcal{E}(\mathcal{A})} (q_2, R_2, R'_2)$ *and denote* $d(v, v') = \max |v_i - v'_i|$ *(i.e. the max-distance), and also* $d(v, R) = \min\{d(v, v') \mid v' \in R\}$.

Then for all $\Delta > 0$ *and* $0 < \eta < \Delta$ *there exists* $\zeta \leq \eta$ *such that for all* $v_2 \in R_2$ *for which* $d(v_2, R'_2) < \zeta$ *there exists* $v_1 \in R_1$ *for which* $d(v_1, R'_1) < \eta$ *and with* $(q_1, v_1) \xrightarrow{t}_{\Delta} (q_2, v_2)$ *for some* $t \in \mathbb{R}_{>0}$.

The proof is by induction on the number of transitions in $\mathcal{E}(\mathcal{A})$. There are 6 base cases, according to the types of $\mathcal{E}(\mathcal{A})$-transition. The case $(q, R_1, R'_1) \xrightarrow{\circlearrowleft}_{\mathcal{E}(\mathcal{A})} (q, R_2, R'_2)$ relies on the following easy generalization of Theorem 7.3 of [Pur98]:

Lemma 2. *Given* $R \in \mathsf{Reg}_{\mathcal{A}}$ *and* $q \in Q$ *with* $((q, R), (q, R)) \in \delta^+_{\mathcal{R}}$, *then for any* $v, v' \in R$ *and for any* $\Delta > 0$, $(q, v') \in \mathsf{Reach}_{\Delta}(q, v)$.

The following straightforward corollary of Lemma 2 will be essential in the construction of our symbolic algorithm:

Corollary 1. *Suppose that the state regions* $(q_1,R_1)\neq(q_2,R_2)$ *belong to the same strongly connected component in the region automaton, and* $((q_2,R_2),(q_3,R_3))\in\delta^*_{\mathcal{R}}$. *Then for any* $v_1\in R_1$, $v_3\in R_3$ *and for any* $\Delta>0$, $(q_3,v_3)\in\mathsf{Reach}_\Delta(q_3,v_3)$.

Note that the validity of this corollary relies on the fact that we only consider bounded regions.

The proof of the direct inclusion in Theorem 1 relies on a "continuous" presentation of trajectories. In the sequel, for a mapping $f:A\to B\times C$, $f\big|_2:A\to C$ denotes the second projection. Also, for a real function $f:I\to A$ with $I\subseteq\mathbb{R}_{\geq 0}$ and $J\subseteq I$, $f\big|_J$ is the usual restriction of f to J.

Definition 2. *A* ***continuous*** Δ***-trajectory*** ($\Delta\in\mathbb{R}_{\geq 0}$) *is a mapping* $\phi:[0,\alpha[\to Q\times\mathbb{R}^n_{\geq 0}$ *satisfying the following properties:*

1. *For each* $q\in Q$, $\phi^{-1}(q\times\mathbb{R}^n_{\geq 0})$ *is a finite union of left-closed, right-open intervals* $\left(I^i_{\phi,q}\right)_{1\leq i\leq n_{\phi,q}}$, *with* $I^i_{\phi,q}=[\alpha^i_q,\alpha^{i+1}_q[$.
2. *For any two distinct states* $q,r\in Q, q\neq r$, *and any two adjacent intervals* $I^i_{\phi,q},I^j_{\phi,r}$ *(i.e.,* $\alpha^{i+1}_q=\alpha^j_r$*), there exists a transition* $(q,C,X,r)\in\delta$ *which creates the "jump" from* $I^i_{\phi,q}$ *to* $I^j_{\phi,r}$ *in the following sense: if we denote* $v=\lim\limits_{x\nearrow\alpha^{i+1}_q}\phi\big|_2(x)$ *and* $v'=\phi\big|_2(\alpha^j_r)$, *then* $v\models C$ *and* $v'=v[X:=0]$.
3. *For each* $q\in Q$ *for which* $n_{\phi,q}>0$, *for each* $1\leq i\leq n_{\phi,q}$ *and each* $t,t'\in I^i_{\phi,q}$ *with* $t<t'$, *there exists* $u\in\mathbf{B}^n$ *such that* $\phi\big|_2(t')=\phi\big|_2(t)+(t'-t)(\mathbf{1}+\Delta u)$.

The continuous Δ*-trajectory* ϕ *is* ***canonical*** *if the following property holds:*

4. *For each* $t,t'\in[0,\alpha[$, *if there exists* $R\in\mathsf{Reg}_{\mathcal{A}}$ *and* $q\in Q$ *such that* $\phi(t),\phi(t')\in\{q\}\times R$ *and for all* t'' *with* $t\leq t''\leq t'$, $\phi(t'')\in\{q\}\times\mathbb{R}^n_{\geq 0}$, *then for all* $t\leq t''\leq t'$, $\phi(t'')\in\{q\}\times R$.

The second property holds due to the assumption which forbids taking two discrete transitions without letting time pass. The fourth also is consistent since we only consider automata without self loops.

Canonical continuous trajectories avoid "volutes" between $\mathtt{t}$-neighbors. The following property shows that each Δ-trajectory, which gives only "essential points" through the behavior of a system, can be associated with a canonical continuous trajectory, which in fact completes the Δ-trajectory with all the intermediary points:

Proposition 3. *For each* Δ*-trajectory* $\tau=\left((q_{i-1},v_{i-1})\xrightarrow{\xi_i}(q_i,v_i)\right)_{1\leq i\leq k}$ ($\Delta\geq 0$) *there exists a canonical continuous* Δ*-trajectory* $\phi:[0,\alpha[\to Q\times\mathbb{R}^n_{\geq 0}$ *for which* $\phi(0)=(q_0,v_0)$, $\lim_{t\nearrow\alpha}\phi(t)=(q_k,v_k)$, *and for each* $1\leq i<k$ *there exists* α_i *with* $\alpha_{i-1}<\alpha_i$ *such that* $\phi(\alpha_i)=(q_i,v_i)$.

The proof follows by easy induction on the the length k of τ.

The following property states that, if we "simulate" the behavior of a Δ-trajectory ϕ with a "pseudo"-0-trajectory ϕ', then the final points in the two trajectories cannot be "too far" one from the other. In some sense, the simulating pseudo-0-trajectory models what would happen in $\overline{\mathcal{A}}$, if we were to "follow" the same run that is associated with ϕ, take the transitions at the same time points but without any clock drift and without checking any guard on the transitions (i.e. just resetting clocks).

Proposition 4. *Given a canonical continuous Δ-trajectory $\phi : [0, \alpha[\to Q \times \mathbb{R}^n_{\geq 0}$, consider a mapping $\phi' : [0, \alpha[\to Q \times \mathbb{R}^n_{\geq 0}$ with $\phi'(0) = \phi(0)$ and satisfying the following properties:*

1. *For all $q \in Q, 1 \leq i \leq n_{\phi,q}, t, t' \in I^i_{\phi,q}$ with $t < t'$, $\phi'(t') = \phi'(t) + t' - t$.*
2. *For any two states $q, r \in Q, q \neq r$ and any two adjacent intervals $I^i_{\phi,q}, I^j_{\phi,r}$ (with $\alpha^{i+1}_q = \alpha^j_r$), if (q, C, X, r) is the transition for which condition 2 in Definition 2 is met for ϕ, and we denote $v = \lim_{x \nearrow \alpha^{i+1}_q} \phi'\big|_2(x)$ and $v' = \phi'\big|_2(\alpha^j_r)$, then $v' = v[X := 0]$.*

Then for each $t \in [0, \alpha[$, $d(\phi\big|_2(t), \phi'\big|_2(t)) \leq t\Delta$.

Recall that d denotes the max-distance. The proof can be again given by induction on the number of regions through which ϕ passes.

Note that, by the construction in Proposition 4, there exists a unique mapping ϕ' associated to ϕ – we will call it the **0-approximation of** ϕ. ϕ' is not really a 0-trajectory since in condition 2 above we may have $v \not\models C$.

For the following lemma, we denote $\xrightarrow{\mathtt{t},0}$ the union of the identity relation with the immediate successor relation in $\mathcal{R}_{\mathcal{A}}$. The result here is needed when showing that regions that are "arbitrarily close" are forward-propagated through the same types of region transitions, then we obtain also regions that are "arbitrarily close":

Lemma 3. *Consider two tuples of regions $R_1, \ldots, R_n, R'_1, \ldots R'_n$ and two extra regions R, R' such that $V(R) = \bigcap_{1 \leq i \leq n} V(R_i)$ and $V(R') = \bigcap_{1 \leq i \leq n} V(R'_i)$.*

1. *Suppose that $(q, R_i) \xrightarrow{\mathtt{t},0} (q, R'_i)$ for all $1 \leq i \leq n$ and some $q \in Q$. Then $(q, R) \xrightarrow{\mathtt{t},0} (q, R')$.*
2. *If $(q, R_i) \xrightarrow{\downarrow} (q', R'_i)$ for all $1 \leq i \leq n$ and some $q, q' \in Q$, then $(q, R) \xrightarrow{\downarrow} (q', R')$.*

The first result follows by observing how linear combinations of vertices of a region evolve during time steps, whereas the second is straightforward.

The following technical property is needed when proving a somewhat reverse of the previous lemma: if two regions can be reached from one another via some (sufficiently small) time-passage transition with some drift $\Delta > 0$, and they are neighbors of some regions that can be reached from one another in the 0-drift region automaton, then, altogether, the four regions form a transition in $\mathcal{E}(\mathcal{A})$:

Proposition 5. *Given $\Delta > 0$, two regions $R_1 \neq R_2$, and a (canonical) continuous Δ-trajectory $\phi : [0, \alpha[\to Q \times \mathbb{R}^n_{\geq 0}$, suppose that $\phi(t) \in \{q\} \times (R_1 \cup R_2)$ for some $q \in Q$ and all $t \in [0, \alpha[$, and also that $\phi(0) \in (q, R_1)$, $\lim_{t \nearrow \alpha} \phi(t) \in (q, R_2)$. Then there exist R'_1, R'_2 such that $V(R'_1) \subseteq V(R_1)$ and $V(R'_2) \subseteq V(R_2)$ such that $(q, R'_1) \xrightarrow{\mathtt{t}} (q, R'_2)$. Moreover, $(q, R_1, R'_1) \to_{\mathcal{E}(\mathcal{A})} (q, R_2, R'_2)$.*

Together, Lemma 3 and Proposition 5 show how the neighborship relation between regions is related with the transition relation in the region automaton.

The final step in the proof of the direct inclusion in Theorem 1 is the following proposition. Here, we denote $T = card(\theta_{\mathcal{E}(\mathcal{A})})$ and $K = 2^{3n+1}$. Also $\mathbf{A}^{K+2}_{T+2} = \frac{(T+2)!}{(T-K)!}$ is the number of ordered tuples of $2^{3n+1} + 2$ elements from a set of $card(\theta_{\mathcal{E}(\mathcal{A})})+2$ elements. Note that $K < T$ for any automaton $\mathcal{A}$.

Proposition 6. *Take $\phi : [0, \alpha[\to Q \times \mathbb{R}^n_{\geq 0}$ a canonical continuous Δ-trajectory, with $\Delta < \frac{1}{2(\mathbf{A}^{K+2}_{T+2})^2+2} \cdot \sqrt{\frac{2}{n}}$. Denote $\big(q_i, R_i\big)_{0 \leq i \leq m_\phi}$ the sequence of state regions to which the points in ϕ belong, that is, if $\phi(t) \in (q_i, R_i), \phi(t') \in (q_{i+1}, R_{i+1})$ then $\forall t'' \in]t, t'[, \phi(t'') \in (q_i, R_i) \cup (q_{i+1}, R_{i+1})$. Then there exist regions $(\tilde{R}_i)_{0 \leq i \leq m_\phi}$ such that $(q_{i-1}, R_{i-1}, \tilde{R}_{i-1}) \to_{\mathcal{E}(\mathcal{A})} (q_i, R_i, \tilde{R}_i)$ for all $1 \leq i \leq m_\phi$.*

The proof works in two steps: first, for $\alpha < 2(\mathbf{A}^{K+2}_{T+2})^2+2$, we may show that there exist regions $(\tilde{R}_i)_{0 \leq i \leq m_\phi}$ such that $(q_{i-1}, R_{i-1}, \tilde{R}_{i-1}) \xrightarrow{\mathtt{t}_1, \mathtt{t}_2, n, r, \downarrow}_{\mathcal{E}(\mathcal{A})} (q_i, R_i, \tilde{R}_i)$ for all $1 \leq i \leq m_\phi$. Then, for the case of $\alpha \geq 2(\mathbf{A}^{K+2}_{T+2})^2 + 2$, we may show that, after most $2(\mathbf{A}^{K+2}_{T+2})^2 + 2$ transitions, the trajectory must pass through the same state region (q, R), which implies that, after the correspoding sequence of $\xrightarrow{\mathtt{t}_1, \mathtt{t}_2, n, r, \downarrow}_{\mathcal{E}(\mathcal{A})}$-transitions (as constructed in the first part), we may insert a $\circlearrowleft$ transition. The whole argument is based on Lemmas 1, 3 (and some extra technical lemmas) and Propositions 1 and 5. The details of this proof can be found in the report [Dim06].

4 Symbolic Computation of $\mathsf{RegReach}_{\Delta \to 0}(Q_0, 0_n)$

The idea behind our symbolic algorithm is to alternate forward reachability, $\mathtt{t}$-neighbor construction and cycle construction, until a fixpoint is reached. The cycle construction takes advantage of the special form of boundary regions in $\mathcal{E}(\mathcal{A})$ that give the possibility to obtain, symbolically, all regions that are neighbors of reachable regions. Our algorithm uses triples of the form (q, Z, Z') where $q \in Q$ and Z, Z' are zones with $Z' = \overline{Z'} \subseteq Z$. The algorithm generates triples (q, Z, Z') for which, for any regions R, R', if $R \subseteq Z$, $R' \subseteq Z'$ and $V(R) \supseteq V(R')$, then $(q_0, \mathbf{0}_n, \mathbf{0}_n) \to_{\mathcal{E}(\mathcal{A})} (q, R, R')$. But let us introduce first some notations.

The term $\mathcal{A}$-*zones* denotes zones $Z \subseteq [0, M_{\mathcal{A}}]^n$ (recall $M_{\mathcal{A}}$ is the maximal constant used in $\mathcal{A}$). The set of $\mathcal{A}$-zones is denoted $\mathcal{Z}_{\mathcal{A}}$. We also say that an $\mathcal{A}$-zone Z is *time passage closed* if for all $v \in Z$ and $q \in Q$, if $(q, v) \xrightarrow{t} (q, v')$ then $v' \in Z$. For any $\mathcal{A}$-zone Z we denote

$$\mathtt{t}\text{–}\mathsf{Nghbr}(Z) = Z \cup \bigcup \{R' \mid R \subseteq Z, R \in \mathsf{Reg}_{\mathcal{A}}, R, R' \text{ are } \mathtt{t}\text{-neighbors}\}$$

We also denote $\mathtt{t}\text{–}\mathsf{Nghbr}(q,Z)=\{q\}\times\mathtt{t}\text{–}\mathsf{Nghbr}(Z)$ for any $q\in Q$.

The following proposition gives a set of properties that characterize the $\mathtt{t}\text{–}\mathsf{Nghbr}$ operator and relate it with the boudary regions in $\mathcal{E}(\mathcal{A})$:

Proposition 7. *1. Suppose Z is an $\mathcal{A}$-zone, with $C_Z=\bigwedge_{0\le j<i\le n}x_i-x_j\in I_{ij}$. Then $\mathtt{t}\text{–}\mathsf{Nghbr}(Z)$ is defined by the constraint in normal form*

$$C_{\mathtt{t}\text{–}\mathsf{Nghbr}(Z)}:\bigwedge\nolimits_{1\le i\le n}(x_i\in I_{i0})\wedge\bigwedge\nolimits_{1\le j<i\le n}\Big(x_i\text{–}x_j\in\big(I_{ij}+\,]\text{–}1,1[\big)\cap[-M_{\mathcal{A}},M_{\mathcal{A}}]\Big)$$

2. Moreover, if Z is time passage closed, then for all $R_1,R_1'\subseteq Z$ with $V(R_1)\supseteq V(R_1')$, $(q,R_1,R_1')\xrightarrow{\mathtt{t}_1,\mathtt{t}_2,r,n}_{\mathcal{E}(\mathcal{A})}(q,R_2,R_2')$ if and only if there exists $(q,Z')\subseteq\mathtt{t}\text{–}\mathsf{Nghbr}(Z)$ for which $R_2,R_2'\subseteq Z'$ and $V(R_2)\supseteq V(R_2')$.

3. Finally, suppose Z_1,Z_2 are two $\mathcal{A}$-zones which are time passage closed. Then $Z_2=\mathtt{t}\text{–}\mathsf{Nghbr}(Z_1)$ if and only if for any region $R_1\subseteq Z_1$ that is $\mathtt{t}$-aligned, and for any region R_2 which is a $\mathtt{t}$-neighbor for R_1, we have that $R_2\subseteq Z_2$.

Given a transition $\tau=(q,X,C,q')$ and a zone Z, the *forward propagation of* (q,Z) *along* τ is:

$$\mathbf{Fwd}(q,Z,\tau)=\big\{(q',Z')\mid\exists v\in Z,v'\in Z',v''\in\mathbb{R}^n_{\ge0},t,t'\in\mathbb{R}_{\ge0}\text{ s.t. }(q,v)\xrightarrow{t}_0(q,v'')\xrightarrow{\downarrow}_0(q',v''[X:=0])\xrightarrow{t'}_0(q',v')\text{ and }v''\models C\big\}$$

The forward propagation of (q,Z) is then

$$\mathbf{Fwd}(q,Z)=\mu X.\big((q,Z)\cup\bigcup\nolimits_{\tau\in\delta}\mathbf{Fwd}(X,\tau)\big)$$

It is well known [Yov98] that, for any zone Z, state q and transition τ,

$\mathbf{Fwd}(q,Z,\tau)$ and $\mathbf{Fwd}(q,Z)$ are computable symbolically if there is no diagonal constraint in the given automaton $\mathcal{A}$ (see [BLR05]) – which is the case here.

We may then see that forward reachability and $\mathtt{t}$-neighborhoodness are related by the following property:

$$\mathtt{t}\text{–}\mathsf{Nghbr}(q,Z)=\bigcup\big\{(q,R)\mid\exists(q,R_1)\xrightarrow{\mathtt{t}}{}^*(q,R_2)\text{ in }\mathcal{R}_{\mathcal{A}},\ \exists R_3\in\mathsf{Reg}_{\mathcal{A}}\text{ s.t. }R_3\subseteq\mathtt{t}\text{–}\mathsf{Nghbr}(R_1)\cap Z,R\subseteq\mathtt{t}\text{–}\mathsf{Nghbr}(R_2)\big\}$$

For each transition $\tau=(q_1,C,X,q_2)\in\delta$, we denote $\overline{\tau}=(q_1,\overline{C},X,q_2)$ the *closure* of τ, with $\overline{C}$ being the closure of C (in the sense that all inequalities in C are transformed into nonstrict). Define then the *forward propagation* of a triple (q,Z,Z') along the transition τ as:

$$\mathbf{FwBnR}(q_1,Z_1,Z_1',\tau):=\big\{(q_2,Z_2,Z_2')\mid(q_2,Z_2')\in\mathbf{Fwd}(q_1,Z_1',\overline{\tau}),\ Z_2\in\mathtt{t}\text{–}\mathsf{Nghbr}(Z_2')\big\}$$

The following proposition shows that the forward propagations of boundary regions can be computed symbolically, using the **FwBnR** operator:

Proposition 8. *Given $q_1 \in Q$ and Z_1, Z_1' two $\mathcal{A}$-zones which are time passage closed and with $Z_1 = \mathtt{t}\text{–}\mathsf{Nghbr}(Z_1')$. Then $(q_2, R_2, R_2') \in \mathbf{FwBnR}(q_1, Z_1, Z_1', \tau)$ if and only if there exist $(q_1, R_1, R_1') \in \mathcal{Q}_{\mathcal{E}(\mathcal{A})}$ with $R_1 \subseteq Z_1$ $R_1' \subseteq Z_1'$, and $(q_3, R_3, R_3'), (q_4, R_4, R_4') \in \mathcal{Q}_{\mathcal{E}(\mathcal{A})}$ such that*

$$(q_1, R_1, R_1') \xrightarrow{\mathtt{t}_1,\mathtt{t}_2,n,r}_{\mathcal{E}(\mathcal{A})} (q_3, R_3, R_3') \xrightarrow{\downarrow}_{\mathcal{E}(\mathcal{A})} (q_4, R_4, R_4') \xrightarrow{\mathtt{t}_1,\mathtt{t}_2,n,r} (q_2, R_2, R_2')$$

where the middle transition is associated with τ.

In the sequel, we will consider each set of tuplpes $\mathcal{Z} \subseteq Q \times \mathcal{Z}_{\mathcal{A}}$ as the set of state regions belonging to $\mathcal{Z}$. Hence, we abuse notation and denote $\mathcal{Z}_1 = \mathcal{Z}_2$ if $\{(q, R) \mid \exists (q, Z) \in \mathcal{Z}_1, R \subseteq Z\} = \{(q, R) \mid \exists (q, Z) \in \mathcal{Z}_2, R \subseteq Z\}$.

For each set of triples $\Upsilon \subseteq Q \times \mathcal{Z}_{\mathcal{A}} \times \mathcal{Z}_{\mathcal{A}}$, we define $\mathbf{FwBnR}(\Upsilon)$ as the set of triples (q, Z, Z') which can be reached from Υ by repeatedly applying forward propagation steps,

$$\mathbf{FwBnR}(\Upsilon) = \mu X.\big(\Upsilon \cup \bigcup\nolimits_{\tau \in \delta} \bigcup\nolimits_{(q,Z,Z') \in X} \mathbf{FwBnR}(q, Z, Z', \tau)\big)$$

This set can be computed as a fixpoint: $\mathbf{FwBnR}(\Upsilon) = \bigcup_{n \in \mathbb{N}} F_n$ where $F_0 = \Upsilon$ and $F_{n+1} = \bigcup_{\tau \in \delta} \bigcup_{(q,Z,Z') \in F_n} \mathbf{FwBnR}(q, Z, Z', \tau)$.

On the other hand, for each set $\mathcal{Z} \subseteq Q \times \mathcal{Z}_{\mathcal{A}}$, we define $\mathbf{Cyc}(\mathcal{Z})$ as the subset of $\mathcal{Z}$ that contains only regions which lie on a cycle in $\mathsf{Reg}_{\mathcal{A}}$,

$$\mathbf{Cyc}(\mathcal{Z}) = \bigcup \big\{(q, Z) \subseteq \mathcal{Z} \mid \forall R \subseteq Z, R \in \mathcal{R}_{\mathcal{A}}, ((q, R), (q, R)) \in \delta_{\mathcal{R}}^{+}\big\}$$

Proposition 9. $\mathbf{Fwd}(\mathbf{Cyc}(\mathcal{Z})) = \mathbf{Fwd}\Big(\nu X.\big(\mathcal{Z} \cap \mathbf{Fwd}(X)\big)\Big)$.

This result is a corollary of a property related to strongly connected components (s.c.c.) in general graphs, that we give in the following:

Lemma 4. *Consider a finite graph $G = (V, E)$ ($E \subseteq V \times V$, $card(V) < \infty$) in which $E = E_1 \cup E_2$ with $E_1 \cap E_2 = \emptyset$ and such that E does not contain self loops. Denote E^*, E_1^* and E_2^* the reflexive-transitive closure of E, resp. E_1, E_2, and for any $U \subseteq V$, define as usual $E(U) = \big\{v \in V \mid \exists u \in U, (u, v) \in E\big\}$ and $E^*(U) = \mu X.(U \cup E(X))$. Also denote:*

$$\mathbf{Fwd}(U) = \big\{v \in V \mid \exists u \in U, v_1, v_2 \in V, (u, v_1), (v_2, v) \in E_2^*, (v_1, v_2) \in E_1\big\}$$
$$\mathsf{SCC}_{\geq 2}(U) = \bigcup \big\{W \subseteq U \mid W \text{ is a s.c.c. with } card(W) \geq 2\big\}$$

Suppose that $U = E^*(U)$ and there are no nontrivial cycles containing only edges from E_2. Then

$$E^*(\mathsf{SCC}_{\geq 2}(U)) = E^*\Big(\nu X.(U \cap \mathbf{Fwd}(X)\Big)$$

Proof. For the left-to-right inclusion, take $v \in \mathsf{SCC}_{\geq 2}(U)$, which means that there exists a s.c.c. $W \subseteq U$ with $card(W) \geq 2$ and $v \in W$. Note first that

$U = E^*(U)$ implies $\mathbf{Fwd}(E^*(W)) \subseteq U$. On the other hand, due to the fact that E_2 contains no nontrivial cycles, we must have that $\mathbf{Fwd}(E^*(W)) = E^*(W)$. This means that $E^*(W)$ is a fixpoint for the mapping $X \mapsto U \cap \mathbf{Fwd}(X)$, and hence $v \in W \subseteq \nu X.(U \cap \mathbf{Fwd}(X))$ which is the greatest fixpoint.

For the reverse proof, take W with $W = U \cap \mathbf{Fwd}(W)$ and pick some $v \in W$. The fixpoint equation implies that for any k there exists a sequence of vertices $v_1, \ldots, v_k \in W$ such that $(v_i, v_{i+1}) \in E$ $(1 \leq i \leq k)$ with $v_{k+1} = v$. Also $v_i \neq v_{i+1}$, since E contains no self loops. By finiteness of V, there exist $1 \leq j_1 < j_2 - 1 \leq k$ with $v_{j_1} = v_{j_2}$. It follows that there exists a s.c.c. $W' \subseteq W$ with $v_{j_1}, \ldots, v_{j_2-1} \in W'$ and also $card(W') \geq 2$. But this implies that $v \in E^*(W') \subseteq E^*(\mathsf{SCC}_{\geq 2}(U))$. □

Remark 2. Note that, in general, $\mathsf{SCC}_{\geq 2}(U) \neq \nu X.\big(U \cap \mathbf{Fwd}(X)\big)$.

Proposition 9 is then an easy corollary of this lemma, if we take G as the region graph, with $E_1 = \xrightarrow{\mathtt{t}}$ and $E_2 = \xrightarrow{\downarrow}$. Note that our assumption on the given timed automaton ensure the hypotheses in the lemma. More specifically, the existence of the extra clock which is reset and is checked to be > 0 on each transition implies the hypothesis on E_2. not containing nontrivial cycles.

The construction of $\mathbf{Fwd}(\mathbf{Cyc}(\mathcal{Z}))$ involves the fixpoint computation of the inner greatest fixpoint, as the "limit" of the the sequence $C_0 = \mathcal{Z}$ and $C_{n+1} = C_n \cap \mathbf{Fwd}(C_n)$. Then, when this sequence stabilizes, we apply forward closure.

Our symbolic algorithm for computing $\mathsf{RegReach}_{\Delta \to 0}(q_0, \mathbf{0}_n)$ is the following:

Algorithm 1 Construction of $\mathsf{Reach}_{\mathcal{E}(\mathcal{A})}(q_0, \mathbf{0}_n, \mathbf{0}_n)$

$BReach := \{(q_0, \mathbf{0}_n, \mathbf{0}_n)\}$; $PrevBReach := \emptyset$;
while $BReach \neq PrevBReach$
  $PrevBReach := BReach$;
  $BReach := \mathbf{FwBnR}(BReach)$;
  $\mathcal{Z} := \{(q, Z') \mid \exists Z, (q, Z, Z') \in BReach\}$;
  $BReach := BReach \cup \{(q, Z, Z') \mid (q, Z') \in \mathbf{Fwd}(\mathbf{Cyc}(\mathcal{Z})), Z = \mathtt{t}\text{–}\mathsf{Nghbr}(Z')\}$;
end while ;
return $BReach$;

Theorem 2. *Let $\mathcal{A}$ be a bounded timed automaton with no self loops and in which there exists a clock x such that all transitions (q, C, X, q') have $x \in X$ and $C \wedge (x = 0)$ not satisfiable. Then $(q, R) \in \mathsf{RegReach}_{\Delta \to 0}(q_0, \mathbf{0}_n)$ if and only if, at the end of the above algorithm, there exists $(q, Z, Z') \in BReach$ such that $R \subseteq Z$.*

This result is a corollary of Proposition 8 and of Theorem 1. Note also that Corollary 1 is essential in the proof of this theorem, as it allows considering forward propagations of strongly connected components, instead of just cycles in the region graph.

Comparison with [DK06]. The symbolic construction in the last algorithm is different from the one of [DK06]: in that paper, a *stable zone* W_σ is constructed,

symbolically, *for each cycle* σ in the timed automaton. The fixpoint definition for W_σ is $W_\sigma = \nu X.(\mathbf{Fwd}(X) \cap \mathbf{Bck}(X))$. Hence, a preliminary analysis of the graph of the timed automaton is needed, in which all the cycles of the graph have to be constructed. It is well-known that the number of cycles in a graph is superexponential in the size of the graph, hence, at least in theory, the approach of [DK06] may lead to a superexponential time complexity.

One might also ask whether the approach from [DK06] may apply to a strongly connected component rather than to only one cycle at a time. The answer is negative in general, for the following reasons: first, as already [DK06] note, W_σ is in general larger than the set of regions which lie on a cycle in the region graph that is "induced" by σ. However, any region in W_σ is Δ-reachable (for any Δ) from any other region due to a convexity argument regulating the Δ-trajectories through the cycle σ. This argument no longer applies for strongly connected components with more than one cycle. As a counterexample, in the timed automaton in Figure 3, if we put $\mathcal{Z} = \{(q_2, R) \mid R \in \mathcal{R}_\mathcal{A}\}$, then

$$W = \mathcal{Z} \cap \nu X.(\mathbf{Fwd}(X) \cap \mathbf{Bck}(X)) = \{(q_2, (x \in [0,3]) \vee (x \in [4,6]))\}$$

which is not convex (here **Bck** is the backward propagation of a set of zones). And it should be clear that, from the state region $(q_2, x \in [4,5])$, no region within the $\mathcal{A}$-zone $(q_2, x \in [0,3])$ can be Δ-reached. More generally, reaching some region within W does not give guaranty that all W is Δ-reachable.

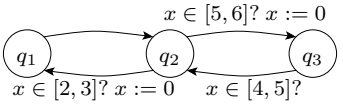


Fig. 3. An example of a non-convex generalization of "stable sets" of [DK06].

5 Conclusions

We have presented a construction for solving the following safe implementation problem: given a timed automaton $\mathcal{A}$ and some n-dimensional zone Z, does there exist a clock drift Δ for which no trajectory in $\mathcal{A}$ in which clocks drifts with at most Δ units reaches Z? The construction generalizes [Pur98], by allowing also the handling of non-closed constraints. We also give a symbolic algorithm that builds the set of zones that are reachable with arbitrarily small clock drifts.

Our algorithm works by constructing, symbolically, forward propagations of strongly connected components in the region graph. Most of the constructions in our algorithm work also with representations of *sets of* zones, like the clock decision diagrams of [LPWY99]. The only construction that could raise problems is $\mathsf{t{-}Nghbr}$, which, as defined, can only be applied to one DBM at a time. We are interested in finding ways to bypass this problem for constructing an algorithm which is fully compatible with CDDs.

On the other hand, our technique of considering strongly connected components instead of just cycles in the region graph can be easily applied to automata

containing only closed constraints, as in [Pur98, DK06]. It is possible that, in that setting, the above compatibility problem between t−Nghbr and CDDs be solvable in a easier way, since in the closed constraints case, region neighborhoodness means regions with nonempty intersection.

Up to the author's knowledge, there exist no algorithms allowing the symbolic computation of the non-trivial strongly connected components in a graph, employing only the set-based constructions that are used in reachability algorithms for timed systems – that is, union, intersection, forward or backward propagation. Symbolic algorithms with good complexity like [GPP03, BGS00] use a "pick" function which returns a single node in the graph, and employ set difference. First, picking a region in a zone, though not an expensive operation, might prove to be a harmful operation w.r.t. set-based structures like clock-difference diagrams. Secondly, set difference, in our setting, amounts to DBM subtraction, which is known not to be a "nice" operation on DBMs. Some heuristics for DBM subtraction have been investigated in [DHLP06].

References

[AD94] Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)

[AHV93] Alur, R., Henzinger, T.A., Vardi, M.: Parametric real-time reasoning. In: Proceedings of STOC'93, pp. 592–601. ACM Press, New York (1993)

[AT05] Altisen, K., Tripakis, S.: Implementation of timed automata: An issue of semantics or modeling? In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 273–288. Springer, Heidelberg (2005)

[ATP01] Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)

[BDM+98] 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)

[BGS00] Bloem, R., Gabow, H.N., Somenzi, F.: An algorithm for strongly connected component analysis in n log n symbolic steps. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 37–54. Springer, Heidelberg (2000)

[BLR05] 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)

[DDMR04] DeWulf, M., Doyen, L., Markey, N., Raskin, J.F.: Robustness and implementability of timed automata. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 118–133. Springer, Heidelberg (2004)

[DDR05a] DeWulf, M., Doyen, L., Raskin, J.-F.: Systematic implementation of real-time models. In: Fitzgerald, J.A., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 139–156. Springer, Heidelberg (2005)

[DDR05b] DeWulf, M., Doyen, L., Raskin, J.F.: Almost asap semantics: from timed models to timed implementations. Formal Aspects of Computing 17(3), 319–341 (2005)

[DHLP06] David, A., Hakanson, J., Larsen, K.G., Petersson, P.: Model checking timed automata with priorities with DBM subtraction. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 128–142. Springer, Heidelberg (2006)

[Dim06] Dima, C.: Dynamical properties of timed automata revisited. Technical Report TR-2006-03, LACL, Université Paris 12 (2006)

[DK06] Daws, C., Kordy, P.: Symbolic robustness analysis of timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 143–155. Springer, Heidelberg (2006)

[GPP03] Gentilini, R., Piazza, C., Policriti, A.: Computing strongly connected components in a linear number of symbolic steps. In: Proceedings of SODA'03, pp. 573–582 (2003)

[HHWT97] Henzinger, T.A., Ho, P.-H., Wong-Toi, H.: HYTECH: A model checker for hybrid systems. Software Tools for Technol. Transfer 1, 110–122 (1997)

[HKPV98] Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata. J. Comput. Syst. Sci. 57, 94–124 (1998)

[LPWY99] Larsen, K.G., Pearson, J., Weise, C., Yi, W.: Clock difference diagrams. Nord. J. Comput. 6, 271–298 (1999)

[LPY97] Larsen, K.G., Petterson, P., Yi, W.: Uppaal: Status & developments. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 456–459. Springer, Heidelberg (1997)

[Pur98] 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)

[WT97] Wong-Toi, H.: Analysis of slope-parametric rectangular automata. In: Antsaklis, P.J., Kohn, W., Lemmon, M.D., Nerode, A., Sastry, S.S. (eds.) Hybrid Systems V. LNCS, vol. 1567, pp. 390–413. Springer, Heidelberg (1999)

[Yov98] Yovine, S.: Model-checking timed automata. In: Rozenberg, G. (ed.) Lectures on Embedded Systems. LNCS, vol. 1494, pp. 114–152. Springer, Heidelberg (1998)

Robust Sampling for MITL Specifications

Georgios E. Fainekos and George J. Pappas

School of Engineering and Applied Science, University of Pennsylvania
{fainekos,pappasg}@seas.upenn.edu

Abstract. Real-time temporal logic reasoning about trajectories of physical systems necessitates models of time which are continuous. However, discrete time temporal logic reasoning is computationally more efficient than continuous time. Moreover, in a number of engineering applications only discrete time models are available for analysis. In this paper, we introduce a framework for testing MITL specifications on continuous time signals using only discrete time analysis. The motivating idea behind our approach is that if the dynamics of the signal fulfills certain conditions and the discrete time signal robustly satisfies the MITL specification, then the corresponding continuous time signal should also satisfy the same MITL specification.

1 Introduction

Assume that we would like to test the transient response of an electronic circuit to a predetermined input signal. Since analytical solutions exist only for a few simple cases, the design, verification and validation of such systems still relies heavily on testing the actual circuit or, more commonly, on simulations [1]. In either case, we end up with a discrete time (or sampled) representation of the continuous time signal that we have to analyze. On the other hand, the properties of the system that we would like to verify are – in most of the cases – with respect to the continuous time behavior of the system.

In particular, properties like overshoot, rise time, delay time, settling time and other constraints on the output signal [2] can be very naturally captured using Metric Temporal Logic (MTL) with continuous time semantics [3]. A restricted version of MTL, namely the Metric Interval Temporal Logic (MITL) [4], has been shown to be decidable over continuous time models even without the finite variability assumption [5]. Recent advances on the monitoring [6] and on the synthesis of timed automata from MITL formulas [7] make possible the verification of real-time properties over continuous time models, however as mentioned earlier, such a representation is hard to be obtained for systems with complex dynamics [8].

Therefore, we must resort to approaches that test MTL specifications on timed words, i.e., sequences of states paired with their respective time stamps. Such testing methodologies are mainly based on formula rewriting methods [9] or monitors generated from automata [10,11]. But then, one major issue is immediately apparent. The continuous time signals and their corresponding sampled versions do not necessarily satisfy the same MTL formula ϕ.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 147–162, 2007.

In this paper, we derive conditions on the dynamics of the signal and on the sampling function such that MITL reasoning over timed words can be applied to continuous time signals. The main machinery that we employ for this purpose is the computation of the robustness estimate [12] of a sampled signal with respect to an MITL specification ϕ. In this framework, the atomic propositions in a formula label regions in the value space of the signal. Intuitively, the robustness estimate is the minimum distance of a point of the sampled signal to such a region, which if it was to be entered by the sampled signal, the truth value of ϕ would change. Hence, all we need to do is to guarantee that the dynamics of the signal are such that between any two sampled points the actual continuous time signal does not violate the aforementioned distance. The constraints on the sampling function play another role. They guarantee that there exist enough sampling points such that the validity of MITL formulas is maintained between the two different semantics [13].

Our theoretical results are demonstrated through some examples that indicate the range of systems that the method can be applied to. Even though our analysis holds for signals of infinite duration, we focus our attention to signals of finite duration. This is so, because the analysis of the asymptotic properties of physical systems is a mature research area [8], while the analysis of the transient properties has not received much attention.

2 Temporal Logics and Continuous Time Signals

In this section, we define signals over metric spaces and provide a brief overview of the temporal logics that are interpreted over linear time structures. Let $\mathbb{R}$ be the set of real numbers and $\mathbb{N}$ the set of natural numbers. We denote the extended real number line by $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$. In addition, we use pseudo-arithmetic expressions to represent certain subsets of the aforementioned sets. For example, $\mathbb{R}_{\geq 0}$ denotes the subset of the reals whose elements are greater or equal to zero. We let $\mathbb{B} = \{\bot, \top\}$, where $\top$ and $\bot$ are the symbols for the boolean constants *true* and *false* respectively. Given two sets A, B, let B^A define the set of all functions from A to B. That is, for any $f \in B^A$ we have $f : A \to B$. Finally, given a set A, $\mathcal{P}(A)$ denotes its powerset.

2.1 Continuous Time Signals in Metric Spaces

In this paper, we use continuous time signals in order to capture the behavior of real-time or physical systems. For example, the temperature in a room is a signal whose domain is $\mathbb{R}_{\geq 0}$ and its range is $\mathbb{R}$ (which hopefully stays in the $[20, 25]$ Celsius degree range). Considering real-valued signals instead of just Boolean values allows us to reason about how far are two points in that space. For example, a temperature of 25°C is closer to the temperature of 30°C than to 40°C.

Formally, a continuous time *signal* s is a map $s : R \to X$ such that R is the time domain and X is a metric space. When we consider bounded time signals,

as for example in testing algorithms, then $R = [0, r] \subseteq \mathbb{R}_{\geq 0}$ with $r > 0$, otherwise we let $R = \mathbb{R}_{\geq 0}$. In the following, $\mathfrak{S}$ denotes the set of all possible signals, i.e., $\mathfrak{S} = X^R$. We fix R to refer to a time domain as described above. A metric space is a pair (X, d) such that the topology of the set X is induced by a metric d. In this paper, we only use the notions of metric and neighborhood which we define below.

Definition 1 (Metric). *A metric on a set X is a positive function $d : X \times X \to \mathbb{R}_{\geq 0}$, such that the three following properties hold*

1. $\forall x_1, x_2 \in X.\ d(x_1, x_2) = 0 \Leftrightarrow x_1 = x_2$
2. $\forall x_1, x_2 \in X.\ d(x_1, x_2) = d(x_2, x_1)$
3. $\forall x_1, x_2, x_3 \in X.\ d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3)$

Using a metric d, we can define the distance of a point $x \in X$ from a set $S \subseteq X$. Intuitively, this distance is the shortest distance from x to all the points in S. In a similar way, the depth of a point x in a set S is defined to be the shortest distance of x from the boundary of S.

Definition 2 (Distance, Depth [14] §8). *Let $x \in X$ be a point, $S \subseteq X$ be a set and d be a metric on X. Then, we define the*

- *Distance from x to S to be* $\mathbf{dist}_d(x, S) := \inf\{d(x, y) \mid y \in S\}$
- *Depth of x in S to be* $\mathbf{depth}_d(x, S) := \mathbf{dist}_d(x, X \backslash S)$

We should point out that we use the extended definition of supremum and infimum. In other words, the supremum of the empty set is defined to be bottom element of the domain, while the infimum of the empty set is defined to be the top element of the domain. For example, when we reason over $\overline{\mathbb{R}}$, then $\sup \emptyset := -\infty$ and $\inf \emptyset := +\infty$. Also of importance is the notion of an open ball of radius ε centered at a point $x \in X$.

Definition 3 (ε-Ball). *Given a metric d, a radius $\varepsilon > 0$ and a point $x \in X$, the open ε-ball centered at x is defined as $B_d(x, \varepsilon) = \{y \in X \mid d(x, y) < \varepsilon\}$.*

It is easy to verify that if the distance ($\mathbf{dist}_d$) of a point x from a set S is $\varepsilon > 0$, then $B_d(x, \varepsilon) \cap S = \emptyset$. And similarly, if $\mathbf{depth}_d(x, S) = \varepsilon > 0$, then $B_d(x, \varepsilon) \subseteq S$.

2.2 Metric Interval Temporal Logic over Signals

The Metric Temporal Logic (MTL) was introduced in [3] in order to reason about the quantitative timing properties of boolean signals. A decidable, but restricted version of MTL, namely the Metric Interval Temporal Logic (MITL), was presented in [4]. In this section, we review the basics of the propositional MITL over signals.

Definition 4 (Syntax of MITL in Negation Normal Form). *Let $\mathbb{C}$ be the set of truth degree constants, AP be the set of atomic propositions and $\mathcal{I}$ be a non-empty non-singular interval of R. The set $\Phi_{\mathbb{C}}$ of all well-formed formulas (wff) is inductively defined using the following rules:*

- *Terms: All constants $c \in \mathbb{C}$ and propositions p, $\neg p$ for $p \in AP$ are terms.*
- *Formulas: if ϕ_1 and ϕ_2 are terms or formulas, then $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\phi_1 \mathcal{U}_{\mathcal{I}} \phi_2$ and $\phi_1 \mathcal{R}_{\mathcal{I}} \phi_2$ are formulas.*

The atomic propositions in our case label subsets of the set X. In other words, we define an observation map $\mathcal{O} : AP \to \mathcal{P}(X)$ such that for each $p \in AP$ the corresponding set is $\mathcal{O}(p) \subseteq X$. In the above definition, $\mathcal{U}_{\mathcal{I}}$ is the timed *until* operator and $\mathcal{R}_{\mathcal{I}}$ the timed *release* operator. The subscript $\mathcal{I}$ imposes timing constraints on the temporal operators. The interval $\mathcal{I}$ can be open, half-open or closed, bounded or unbounded, but it must be non-empty ($\mathcal{I} \neq \emptyset$) and non-singular ($\mathcal{I} \neq \{t\}$) in order to be in spirit with the definitions in [4]. Moreover, we define the following operations on the timing constraints $\mathcal{I}$ of the temporal operators:

$$t + \mathcal{I} := \{t + t' \mid t' \in \mathcal{I}\} \quad \text{and} \quad t +_R \mathcal{I} := (t + \mathcal{I}) \cap R$$

for any t in R. Sometimes for clarity in the presentation, we replace $\mathcal{I}$ with pseudometric expressions, e.g. $\mathcal{U}_{[0,1]}$ is written as $\mathcal{U}_{\leq 1}$. In the case where $\mathcal{I} = [0, +\infty)$, we remove the subscript $\mathcal{I}$ from the temporal operators, i.e., we just write $\mathcal{U}$, and $\mathcal{R}$.

Metric Interval Temporal Logic (MITL) formulas are interpreted over signals s. In this paper, we define the boolean semantics of MITL formulas using a valuation function $\langle\!\langle \cdot, \cdot \rangle\!\rangle : \Phi_{\mathbb{B}} \times \mathcal{P}(X)^{AP} \to (\mathfrak{S} \times R \to \mathbb{B})$ and we write $\langle\!\langle \phi, \mathcal{O} \rangle\!\rangle(s,t) = \top$ instead of the usual notation $(\mathcal{O}^{-1} \circ s, t) \models \phi$. In this case, we say that the signal s under observation map $\mathcal{O}$ satisfies the formula ϕ at time t. Here, $\circ$ denotes function composition : $(f \circ g)(t) = f(g(t))$ and $\mathcal{O}^{-1} : X \to \mathcal{P}(AP)$ is defined as $\mathcal{O}^{-1}(x) := \{p \in AP \mid x \in \mathcal{O}(p)\}$ for $x \in X$. For brevity, we drop $\mathcal{O}$ from the notation since without loss of generality we can consider it constant throughout this paper. From an application perspective, we are interested in checking whether $\langle\!\langle \phi \rangle\!\rangle(s,0) = \top$. In this case, we refer to s as a *model* of ϕ and we just write $\langle\!\langle \phi \rangle\!\rangle(s) = \top$ for brevity.

Before proceeding to the actual definition of the semantics, we introduce some auxiliary notation. If $(\mathbb{V}, <)$ is a totally ordered set, then we define the binary operators $\sqcup : \mathbb{V} \times \mathbb{V} \to \mathbb{V}$ and $\sqcap : \mathbb{V} \times \mathbb{V} \to \mathbb{V}$ using the supremum and infimum functions as $x \sqcup y := \sup\{x, y\}$ and $x \sqcap y := \inf\{x, y\}$. Also, for any $V \subseteq \mathbb{V}$ we extend the above definitions as follows $\bigsqcup V := \sup V$ and $\bigsqcap V := \inf V$. Again, we use the extended definition of the supremum and infimum, i.e., $\sup \emptyset := \bot$ and $\inf \emptyset := \top$. Recall that if $(\mathbb{V}, \leq)$ is a totally ordered set, then it is *distributive*, i.e., for all $a, b, c \in \mathbb{S}$ it is $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$ and $a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$. Note that the structure $(\mathbb{B}, <)$ is a totally ordered set with $\bot < \top$ and that $(\mathbb{B}, \sqcap, \sqcup, \neg)$ is a boolean algebra with the complementation defined as $\neg\top = \bot$ and $\neg\bot = \top$.

Definition 5 (CT Semantics of MITL). *Let $s \in \mathfrak{S}$, $\mathcal{O} \in \mathcal{P}(X)^{AP}$ and t, t', $t'' \in R$, then the continuous time semantics of any formula $\phi \in \Phi_{\mathbb{B}}$ is defined by*

$$\langle\!\langle \top \rangle\!\rangle(s,t) := \top \qquad \langle\!\langle \bot \rangle\!\rangle(s,t) := \bot$$
$$\langle\!\langle p \rangle\!\rangle(s,t) := K_{\in}(s(t), \mathcal{O}(p)) \qquad \langle\!\langle \neg p \rangle\!\rangle(s,t) := K_{\in}(s(t), X \backslash \mathcal{O}(p))$$

$$\langle\!\langle \phi_1 \vee \phi_2 \rangle\!\rangle(s,t) := \langle\!\langle \phi_1 \rangle\!\rangle(s,t) \sqcup \langle\!\langle \phi_2 \rangle\!\rangle(s,t)$$
$$\langle\!\langle \phi_1 \wedge \phi_2 \rangle\!\rangle(s,t) := \langle\!\langle \phi_1 \rangle\!\rangle(s,t) \sqcap \langle\!\langle \phi_2 \rangle\!\rangle(s,t)$$
$$\langle\!\langle \phi_1 \mathcal{U}_{\mathcal{I}} \phi_2 \rangle\!\rangle(s,t) := \bigsqcup_{t' \in (t +_R \mathcal{I})} \left(\langle\!\langle \phi_2 \rangle\!\rangle(s,t') \sqcap \bigsqcap_{t \leq t'' < t'} \langle\!\langle \phi_1 \rangle\!\rangle(s,t'') \right)$$
$$\langle\!\langle \phi_1 \mathcal{R}_{\mathcal{I}} \phi_2 \rangle\!\rangle(s,t) := \bigsqcap_{t' \in (t +_R \mathcal{I})} \left(\langle\!\langle \phi_2 \rangle\!\rangle(s,t') \sqcup \bigsqcup_{t \leq t'' < t'} \langle\!\langle \phi_1 \rangle\!\rangle(s,t'') \right)$$

In the above definition, $K_\in$ is the characteristic function of the $\in$ relation, i.e., $K_\in(a, A) = \top$ if $a \in A$ and $\bot$ otherwise. Informally, the formula $\phi_1 \mathcal{U}_{\mathcal{I}} \phi_2$ expresses the property that over the signal s there exists some time in the interval $\mathcal{I}$ that makes ϕ_2 true and, furthermore, for all previous times s satisfies ϕ_1. Intuitively, the release operator $\phi_1 \mathcal{R}_{\mathcal{I}} \phi_2$ states that ϕ_2 should always hold during the time interval $\mathcal{I}$, a requirement which is released when ϕ_1 becomes true. We can also define the temporal operators *eventually* $\Diamond_{\mathcal{I}} \phi = \top \mathcal{U}_{\mathcal{I}} \phi$ and *always* $\Box_{\mathcal{I}} \phi = \bot \mathcal{R}_{\mathcal{I}} \phi$.

3 Temporal Logics and Discrete Time Signals

Even though MITL is decidable over continuous time Boolean signals [5], there do not exist efficient decision procedures as it is the case for the discrete untimed systems [15]. Matters become even worse when we consider hybrid systems with real time requirements and states that evolve in metric spaces. For such systems, a discrete time representation of their continuous time behavior can provide a valuable tool for analysis.

3.1 Sampled Signals

A sampled (or discrete time) signal can represent computer simulated trajectories of physical models or the sampling process that takes place when we digitally monitor physical systems. In the following, we assume that a continuous time signal s is a mathematical object that represents the behavior of a physical system and we define a *sampling function* $\tau \in R^N$ which returns the point in time at which the i-th sample was taken. The set of all sampling functions is denoted by $\mathfrak{T}$, i.e., $\mathfrak{T} = R^N$. We fix $N \subseteq \mathbb{N}$ to be the set indexes for the sampled points. In other words, the discrete time signal $\hat{s} = s \circ \tau$ corresponds to the observable (discretized) behavior of the physical system (see Fig. 1). Two necessary assumptions on any sampling function are : (i) τ must be a monotonic function, i.e., $\tau(i) < \tau(j)$ for $i < j$ and (ii) if R is unbounded then $N = \mathbb{N}$. Notice that the pair $(\mathcal{O}^{-1} \circ \hat{s}, \tau)$ is actually a *timed state sequence*, which is a widely accepted model for reasoning about real time systems [16].

3.2 Metric Interval Temporal Logic over Sampled Signals

We proceed on to define MITL semantics over discrete time signals. Here, we slightly deviate from the usual definition of the semantics over timed state

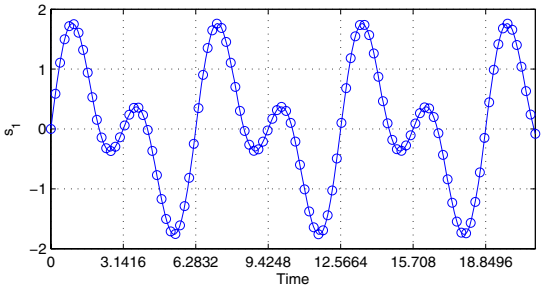


Fig. 1. A continuous time signal $s_1(t) = \sin t + \sin 2t$ (*solid line*) and the corresponding sampled signal $\hat{s}_1$ (*circles*) generated using a constant sampling step of 0.2

sequences. We consider as a model of ϕ the actual continuous time signal s under the sampling function τ. This will enable us to reason about all the continuous time signals which have the same sampled representation in a transparent way. Again, the MITL semantics is defined using a valuation function which now also depends on the sampling function $\tau \in \mathfrak{T}$. We write $\langle\!\langle \phi \rangle\!\rangle_\tau(s,i) = \top$ when the signal s under sampling function τ satisfies the formula ϕ at sample i (as before, the observation map $\mathcal{O}$ is implied). Similarly to the continuous time case, when $i = 0$ and the formula evaluates to $\top$, then we refer to s as a *model* of ϕ under the sampling function τ and we write $\langle\!\langle \phi \rangle\!\rangle_\tau(s) = \top$. In the definition below, we also use the following notation : for $Q \subseteq R$, the *preimage* of Q under τ is defined as : $\tau^{-1}(Q) := \{i \in N \mid \tau(i) \in Q\}$. Notice that $N = \tau^{-1}(R)$.

Definition 6 (DT Semantics of MITL). *Let $s \in \mathfrak{S}$, $\tau \in \mathfrak{T}$, $\mathcal{O} \in \mathcal{P}(X)^{AP}$ and $i, j, k \in N$, then the discrete time semantics of any formula $\phi \in \Phi_{\mathbb{B}}$ is defined by*

$$\langle\!\langle \top \rangle\!\rangle_\tau(s,i) := \top \qquad \langle\!\langle \bot \rangle\!\rangle_\tau(s,i) := \bot$$
$$\langle\!\langle p \rangle\!\rangle_\tau(s,i) := K_\in(\hat{s}(i), \mathcal{O}(p)) \qquad \langle\!\langle \neg p \rangle\!\rangle_\tau(s,i) := K_\in(\hat{s}(i), X \backslash \mathcal{O}(p))$$
$$\langle\!\langle \phi_1 \vee \phi_2 \rangle\!\rangle_\tau(s,i) := \langle\!\langle \phi_1 \rangle\!\rangle_\tau(s,i) \sqcup \langle\!\langle \phi_2 \rangle\!\rangle_\tau(s,i)$$
$$\langle\!\langle \phi_1 \wedge \phi_2 \rangle\!\rangle_\tau(s,i) := \langle\!\langle \phi_1 \rangle\!\rangle_\tau(s,i) \sqcap \langle\!\langle \phi_2 \rangle\!\rangle_\tau(s,i)$$
$$\langle\!\langle \phi_1 \, \mathcal{U}_\mathcal{I} \phi_2 \rangle\!\rangle_\tau(s,i) := \bigsqcup_{j \in \tau^{-1}(\tau(i) +_R \mathcal{I})} \Big(\langle\!\langle \phi_2 \rangle\!\rangle_\tau(s,j) \sqcap \prod_{i \le k < j} \langle\!\langle \phi_1 \rangle\!\rangle_\tau(s,k) \Big)$$
$$\langle\!\langle \phi_1 \, \mathcal{R}_\mathcal{I} \phi_2 \rangle\!\rangle_\tau(s,i) := \prod_{j \in \tau^{-1}(\tau(i) +_R \mathcal{I})} \Big(\langle\!\langle \phi_2 \rangle\!\rangle_\tau(s,j) \sqcup \bigsqcup_{i \le k < j} \langle\!\langle \phi_1 \rangle\!\rangle_\tau(s,k) \Big)$$

4 Robustness Estimate

The main goal of this paper is to derive conditions that guarantee that a signal is a model of an MITL formula with continuous time semantics using only discrete

time reasoning. The main tool that we employ in order to achieve this goal is the *robustness estimate* [12]. In [12], the robustness estimate was used in order to determine neighborhoods of finite timed state sequences that satisfy the same MTL specification. In this paper, the robustness estimate will help us determine a critical distance-threshold that guarantees that a signal satisfies an MITL formula with continuous time semantics.

The robustness estimate can be computed by introducing multi-valued semantics for MITL formulas. In this paper, we differentiate from previous works – see for example [17] – by providing the definition of multi-valued semantics for MITL based on robustness considerations. Let $\mathfrak{R} = (\overline{\mathbb{R}}, \leq)$ be the real line with the usual ordering relation. We propose multi-valued semantics for the Metric Interval Temporal Logic where the valuation function on the predicates takes values over the totally ordered set $\mathfrak{R}$ according to the metric d operating on the state space X of the signal s. For this purpose, we let the valuation function be the depth (or the distance) of the current point of the signal $s \circ \tau(i)$ in a set $\mathcal{O}(p)$ labeled by the atomic proposition p. Intuitively, this distance represents how robustly is the point $s \circ \tau(i)$ within a set $\mathcal{O}(p)$. If this metric is zero, then even the smallest perturbation of the point can drive it inside or outside the set $\mathcal{O}(p)$, dramatically affecting membership.

For the purposes of the following discussion, we use the notation $[\![\phi]\!]$ to denote the robustness estimate with which the signal s satisfies the specification ϕ under the sampling function τ (formally, $[\![\phi]\!]_\tau : \mathfrak{S} \times N \to \overline{\mathbb{R}}$ and, again, the observation map $\mathcal{O}$ is implied).

Definition 7 (Robustness Estimate). *Let $s \in \mathfrak{S}$, $\tau \in \mathfrak{T}$, $c \in \overline{\mathbb{R}}$, $\mathcal{O} \in \mathcal{P}(X)^{AP}$ and $i, j, k \in N$, then the robustness estimate of any formula $\phi \in \Phi_{\mathbb{R} \cup \mathbb{B}}$ with respect to s under the sampling function τ is recursively defined as follows*

$$
\begin{aligned}
[\![\top]\!]_\tau(s,i) &:= +\infty & [\![\bot]\!]_\tau(s,i) &:= -\infty \\
[\![p]\!]_\tau(s,i) &:= \mathbf{depth}_d(\hat{s}(i), \mathcal{O}(p)) & [\![\neg p]\!]_\tau(s,i) &:= \mathbf{dist}_d(\hat{s}(i), \mathcal{O}(p)) \\
[\![c]\!]_\tau(s,i) &:= c \\
[\![\phi_1 \vee \phi_2]\!]_\tau(s,i) &:= [\![\phi_1]\!]_\tau(s,i) \sqcup [\![\phi_2]\!]_\tau(s,i) \\
[\![\phi_1 \wedge \phi_2]\!]_\tau(s,i) &:= [\![\phi_1]\!]_\tau(s,i) \sqcap [\![\phi_2]\!]_\tau(s,i) \\
[\![\phi_1 \mathcal{U}_\mathcal{I} \phi_2]\!]_\tau(s,i) &:= \bigsqcup_{j \in \tau^{-1}(\tau(i)+\mathcal{I})} \Big([\![\phi_2]\!]_\tau(s,j) \sqcap \bigsqcap_{i \leq k < j} [\![\phi_1]\!]_\tau(s,k) \Big) \\
[\![\phi_1 \mathcal{R}_\mathcal{I} \phi_2]\!]_\tau(s,i) &:= \bigsqcap_{j \in \tau^{-1}(\tau(i)+\mathcal{I})} \Big([\![\phi_2]\!]_\tau(s,j) \sqcup \bigsqcup_{i \leq k < j} [\![\phi_1]\!]_\tau(s,k) \Big)
\end{aligned}
$$

It is easy to verify that the semantics of the negation operator give us all the usual nice properties such as the *De Morgan laws*: $a \sqcup b = -(-a \sqcap -b)$ and $a \sqcap b = -(-a \sqcup -b)$, *involution*: $-(-a) = a$ and *antisymmetry*: $a \leq b$ iff $-a \geq -b$ for $a, b \in \overline{\mathbb{R}}$.

5 Continuous Time Satisfiability by Discrete Reasoning

To this point one question remains unanswered. What is the relationship between $\langle\langle\phi\rangle\rangle(s)$ and $\langle\langle\phi\rangle\rangle_\tau(s)$ for a given MITL formula ϕ, signal s and sampling function τ? This is an important question since a sampling function τ may not just change the satisfiability of a formula ϕ with respect to a signal s, but also the validity of the formula [13]. In this section, we develop conditions for the signals in the set $\mathfrak{S}$ and the sampling function τ which can guarantee the equality $\langle\langle\phi\rangle\rangle_\tau(s) = \langle\langle\phi\rangle\rangle(s)$. In the following, we introduce a sequence of assumptions.

First, we need to derive conservative bounds on the divergence of the value of signal s between two consecutive samples i and $i+1$. We do that by requiring that the state distance between any two points in time is bounded by a positive nondecreasing function $\mathcal{E}$ which depends only on the time difference between these two points.

Assumption 1. *The signals in the set $\mathfrak{S}$ satisfy the following condition:*

$$\forall t, t' \in R \,.\, d(s(t), s(t')) \leq \mathcal{E}(|t - t'|), \tag{1}$$

where $\mathcal{E}$ is a positive nondecreasing function.

Such bounds can be easily derived when a signal is Lipschitz continuous.

Definition 8 (Lipschitz Continuity). *Let (X, d) and (X', d') be two metric spaces. A function $f : X' \to X$ is called Lipschitz continuous if there exists a constant $L_f \geq 0$ such that:*

$$\forall x'_1, x'_2 \in X' . d(f(x'_1), f(x'_2)) \leq L_f d'(x'_1, x'_2). \tag{2}$$

The smallest constant L_f is called Lipschitz constant of the function f.

What we are actually interested in is Lipschitz continuity of a signal s with respect to time:

$$\forall t, t' \in R \,.\, d(s(t), s(t')) \leq L_s |t - t'|. \tag{3}$$

Any signal with bounded time derivative satisfies the above condition. Whenever only a number of values of the signal are available to us, instead of an analytical description, we can use methods from optimization theory in order to estimate a Lipschitz constant for the signal [18]. Moreover, if the signal s is the solution of an ordinary differential equation $\dot{s}(t) = f(s(t))$, where f is Lipschitz continuous with constant L_f, then it is possible to estimate a constant L_s for eq. (3).

Example 1. *Consider an autonomous linear system $\dot{s}(t) = As(t)$, where A is Hurwitz. Then, $\|\dot{s}(t)\| = \|As(t)\| \leq \|A\|\|s(t)\|$, where $\|\cdot\|$ is the Euclidean norm. Consider the customary Lyapunov function for stable linear systems $V(x) = x^T Px$, where P is a symmetric and positive definite matrix [8]. It is easy to see that the Lyapunov level sets $\{x \in X \mid V(x) \leq c\}$ are ellipsoids. Recall that any signal which crosses the surface of such a set always remains in the set. Therefore, the distance of $s(t)$ from the origin is always bounded by the radius*

of the minimum ball that contains the ellipsoid $\{x \in X \mid x^T P x \leq s(0)^T P s(0)\}$. *The matrix* P *determines the shape of the ellipsoids and it can be computed by solving the Lyapunov equation :* $PA + A^T P + I = 0$. *The lengths of the axis of the ellipsoid are given by the square roots of the eigenvalues of the matrix* $P_e = V(s(0))P^{-1}$ *[14]. Let* $\lambda_{\max}(P_e)$ *be the maximum eigenvalue of* P_e, *then* $\|s(t)\| \leq \sqrt{\lambda_{\max}(P_e)}$ *for all* $t \in R$. *Hence,* $\|\dot{s}(t)\| \leq \|A\|\sqrt{\lambda_{\max}(P_e)}$ *and the Lipschitz constant is* $L_s = \|A\|\sqrt{\lambda_{\max}(P_e)}$. *In this case, the Lipschitz constant depends on the initial condition* $s(0)$. □

Notice that the bound on the distance between two values of the signal depends on the sampling function τ. In particular, one parameter of the sampling function that we might wish to control is the *maximum sampling step*:

$$\Delta\tau = \sup_{i \in N_{>0}} \{\tau(i) - \tau(i-1)\}. \tag{4}$$

The sampling function τ, i.e., the maximum sampling step $\Delta\tau$, must be such that the relationship between valid formulas in continuous and sampled semantics is maintained [13]. For example, it is easy to see that the formula $\Box_{[1,2]}p$ is true for any signal s if there is no sample in the interval $[1, 2]$. In order to avoid such situations, we must impose certain constraints to $\Delta\tau$. But first, a slight modification of the timing constraints of the temporal operators is required.

Similarly to [19], we strengthen MITL formulas by changing the timing requirements of a given formula ϕ. In detail, we introduce a function $\mathcal{H} : \Phi_{\mathbb{B}} \to \Phi_{\mathbb{B}}$ that recursively operates on a formula ϕ and strengthens the timing constraints as follows:

$$\begin{array}{ll} \mathcal{H}(p) = p & \mathcal{H}(\neg p) = \neg p \\ \mathcal{H}(\phi_1 \vee \phi_2) = \mathcal{H}(\phi_1) \vee \mathcal{H}(\phi_2) & \mathcal{H}(\phi_1 \wedge \phi_2) = \mathcal{H}(\phi_1) \wedge \mathcal{H}(\phi_2) \\ \mathcal{H}(\phi_1 \mathcal{U}_{\mathcal{I}} \phi_2) = \mathcal{H}(\phi_1) \mathcal{U}_{C(\mathcal{I},\Delta\tau)} \mathcal{H}(\phi_2) & \mathcal{H}(\phi_1 \mathcal{R}_{\mathcal{I}} \phi_2) = \mathcal{H}(\phi_1) \mathcal{R}_{E(\mathcal{I},\Delta\tau)} \mathcal{H}(\phi_2) \end{array}$$

where $C(\mathcal{I}, \delta) = \{r \in R \mid cl(B_d(r, \delta)) \subseteq \mathcal{I}\}$ is the δ-contraction and $E(\mathcal{I}, \delta) = \{r \in R \mid cl(B_d(r, \delta)) \cap \mathcal{I} \neq \emptyset\}$ is the δ-expansion of the interval $\mathcal{I}$. Here, cl denotes the closure of a set. The intuition behind the function $\mathcal{H}$ is that a robust specification with respect to the atomic propositions must also be robust with respect to the timing constraints. The necessity of the robustification of the timing requirements will become apparent in the proof of Theorem 1. For example, in order to determine the Boolean truth value of ϕ_2 in $\phi_1 \mathcal{R}_{\mathcal{I}} \phi_2$ for the whole interval $\mathcal{I}$ in continuous time, we must also consider the first samples after and before the interval $\tau(i) +_R \mathcal{I}$.

Remark 1. *The authors in [19] have proven that for any* $\phi \in \Phi_{\mathbb{B}}$, $s \in \mathfrak{S}$, $\tau \in \mathfrak{T}$ *and* $\mathcal{O} \in P(X)^{AP}$, $\langle\!\langle \mathcal{H}(\phi) \rangle\!\rangle_\tau(s, i) = \top$ *implies* $\langle\!\langle \phi \rangle\!\rangle_\tau(s, i) = \top$.

Assumption 2. *The sampling functions in the set* $\mathfrak{T}$ *satisfy the constraint:*

$$\Delta\tau < \min_{\mathcal{I} \in (\mathfrak{I}_{\mathcal{H}(\phi)} \cup \mathfrak{I}_\phi)} \{\sup \mathcal{I} - \inf \mathcal{I}\}. \tag{5}$$

When R *is bounded, the sampling functions in the set* $\mathfrak{T}$ *must also satisfy the constraint :* $\sup R - \tau(\max N) < \Delta\tau$.

In the assumption above, $\mathfrak{I}_\phi$ denotes the set of all the timing constraints $\mathcal{I}$ that appear in the temporal operators of an MITL formula ϕ. Notice that if there exists a singleton interval in the set $\mathfrak{I}_\phi$, then the above assumption cannot be satisfied. This observation justifies the choice of MITL as a specification language instead of MTL. It is easy to see that with respect to the initial formula ϕ, Assumption 2 can be satisfied by the following constraint:

$$\Delta\tau < 1/3 \min_{\mathcal{I}\in\mathfrak{I}_\phi} \{\sup \mathcal{I} - \inf \mathcal{I}\}. \tag{6}$$

Whenever R is a bounded time interval, we have to impose additional constraints on the signal and the MITL formulas. First, we require that all the intervals in $\mathfrak{I}_\phi$ are bounded as it was initially suggested in [6]. This enables us to compute a minimum time $\mathcal{D}(\phi)$ that guarantees in combination with Assumption 2 that there are no subformulas whose truth value was determined by the lack of sampling points. The computation of the minimum time $\mathcal{D}(\phi)$ is performed recursively:

$$\begin{array}{ll} \mathcal{D}(p) := 0 & \mathcal{D}(\neg p) := 0 \\ \mathcal{D}(\phi_1 \vee \phi_2) := \mathcal{D}(\phi_1) \sqcup \mathcal{D}(\phi_2) & \mathcal{D}(\phi_1 \wedge \phi_2) := \mathcal{D}(\phi_1) \sqcup \mathcal{D}(\phi_2) \\ \mathcal{D}(\phi_1 \mathcal{U}_\mathcal{I} \phi_2) := \sup \mathcal{I} + \mathcal{D}(\phi_1) \sqcup \mathcal{D}(\phi_2) & \mathcal{D}(\phi_1 \mathcal{R}_\mathcal{I} \phi_2) := \sup \mathcal{I} + \mathcal{D}(\phi_1) \sqcup \mathcal{D}(\phi_2) \end{array}$$

In particular, we would like to avoid the case where R is a bounded domain and $t + \mathcal{I} \not\subseteq R$. For the shake of example, consider the formula $\Box_{[3,4]} p$ and let the domain of the signal s be $R = [0, 2]$. Then, the formula $\Box_{[3,4]} p$ evaluates to $\top$ simply because $0 +_{[0,2]} [3, 4] = \emptyset$. In order to avert such situations, we must impose one additional constraint (when R is bounded). Namely, for a given formula ϕ and signal s we let $\mathcal{D}(\phi) < \sup R < +\infty$. In other words, both the domain of the signal and all the timing constraints in the formula are bounded from above and below. Now, assume that a temporal subformula $\psi = \psi_1 \mathcal{W}_{\mathcal{I}_k} \psi_2$ of ϕ is at a nesting depth k, where $\mathcal{W} \in \{\mathcal{U}, \mathcal{R}\}$, and let $\{\mathcal{I}_j\}_{j<k}$ be the timing constraints of the temporal operators at lower nesting depths. Informally, the nesting depth of a formula ϕ is defined to be the maximum number of nested temporal operators and it is computed in a similar way to $\mathcal{D}$ where $\sup \mathcal{I}$ is replaced by 1. Then, for all $t \in [0, \sum_{j<k} \sup \mathcal{I}_j]$ we have $t + \mathcal{I}_k \subseteq R$ since $\sum_{j\leq k} \sup \mathcal{I}_j \leq \mathcal{D}(\phi) < \sup R$. Therefore, $t + \mathcal{I}_k = t +_R \mathcal{I}_k$.

Assumption 3. *If the time domain R of the set of signals $\mathfrak{S}$ is bounded, i.e., $\sup R < +\infty$, then for the MITL formula ϕ under consideration it must be $\sup \mathcal{I} < +\infty$ for all $\mathcal{I} \in \mathfrak{I}_\phi$ and, also, $\sup R > \mathcal{D}(\mathcal{H}(\phi))$.*

Lemma 1. *Consider a formula $\phi \in \Phi_\mathbb{B}$ and a sampling function $\tau \in \mathfrak{T}$ and let the Assumptions 2 and 3 hold. Let $\psi = \psi_1 \mathcal{W}_{\mathcal{I}_k} \psi_2$, where $\mathcal{W} \in \{\mathcal{U}, \mathcal{R}\}$, be a subformula of ϕ at nesting depth k and let $\{\mathcal{I}_j\}_{j<k}$ be the timing constraints of the temporal operators at lower nesting depths. If $I = \tau^{-1}(T) \neq \emptyset$, where $T = [0, \sum_{j<k} \sup \mathcal{I}_j]$, then for all $i \in I$ we have $\tau^{-1}(\tau(i) +_R \mathcal{I}_k) \neq \emptyset$.*

Proof. First note that, as mentioned above, by Assumption 3 the set $\tau(i) +_R \mathcal{I}_k$ is equal to $\tau(i) + \mathcal{I}_k$ and, hence, $\tau(i) +_R \mathcal{I}_k \neq \emptyset$. Now, assume that $I = \tau^{-1}(T) \neq \emptyset$.

If both R and $\mathcal{I}_k$ are unbounded, then we immediately get $\tau^{-1}(\tau(i)+\mathcal{I}_k) \neq \emptyset$ for any $i \in I$ since otherwise $N = \tau^{-1}(R)$ would be finite. Assume now that $\mathcal{I}_k$ is bounded and that for some $i \in I$ we get that $\tau^{-1}(\tau(i)+\mathcal{I}_k) = \emptyset$. In other words, we assume that there does not exist $i' \geq i$ such that $\tau(i') \in \tau(i)+\mathcal{I}_k$. Then, the following may hold since $\tau(i)+\mathcal{I}_k$ is an interval of R:

1. for all $i' \in N_{\geq i}$ we have $\tau(i') \prec \inf(\tau(i)+\mathcal{I}_k)$, where $\prec \in \{<, \leq\}$ depending on the constraints of $\mathcal{I}_k$. Note that this can only be the case when R is bounded. Hence, we get that $\sup R - \tau(\max N) \succ \sup R - \inf(\tau(i)+\mathcal{I}_k) \geq \sup(\tau(i)+\mathcal{I}_k) - \inf(\tau(i)+\mathcal{I}_k) \geq \sup \mathcal{I}_k - \inf \mathcal{I}_k > \Delta\tau$, which a contradiction by Assumption 2.
2. there exists $i' \in N_{\geq i}$ such that $\tau(i') \prec \inf(\tau(i)+\mathcal{I}_k)$ and $\sup(\tau(i)+\mathcal{I}_k) \prec \tau(i'+1)$, where $\prec \in \{<, \leq\}$ depending on the constraints of $\mathcal{I}_k$. That is, $\tau(i'+1) - \tau(i') \succ \sup(\tau(i)+\mathcal{I}_k) - \inf(\tau(i)+\mathcal{I}_k) = \sup \mathcal{I}_k - \inf \mathcal{I}_k > \Delta\tau$, which is a contradiction by Assumption 2.

Since $\tau^{-1}(\tau(i) +_R \mathcal{I}_k) \neq \emptyset$, we also get that $\tau^{-1}([0, \sum_{j \leq k} \sup \mathcal{I}_j]) \neq \emptyset$. □

The above assumptions enable us to prove the following theorem which is the main result of this paper.

Theorem 1. *Consider $\phi \in \Phi_{\mathbb{B}}$, $\mathcal{O} \in \mathcal{P}(X)^{AP}$, $s \in \mathfrak{S}$, $\tau \in \mathfrak{T}$ and let Assumptions 1 to 3 hold. Then, $[\![\mathcal{H}(\phi)]\!]_\tau(s,i) > \mathcal{E}(\Delta\tau)$ implies*

$$\forall t \in [\tau(i) - \Delta\tau, \tau(i) + \Delta\tau] \cap R \,.\, \langle\!\langle \phi \rangle\!\rangle(s,t) = \top \tag{7}$$

for any $i \in N$ which satisfies the conditions of Lemma 1.

Proof. The proof of the theorem is by induction on the structure of formula ϕ.

Case $\phi = p \in AP$: $[\![\mathcal{H}(p)]\!]_\tau(s,i) > \mathcal{E}(\Delta\tau)$, i.e., $\mathbf{depth}_d(\hat{s}(i), \mathcal{O}(p)) > \mathcal{E}(\Delta\tau)$. Therefore, $d(\hat{s}(i), x) > \mathcal{E}(\Delta\tau)$ for any $x \in X \backslash \mathcal{O}(p)$. Moreover by Assumption 1, we get that $d(\hat{s}(i), s(t)) \leq \mathcal{E}(\Delta\tau)$ for all $t \in [\tau(i) - \Delta\tau, \tau(i) + \Delta\tau] \cap R$ and $d(\hat{s}(i), s(t)) \leq \mathcal{E}(\Delta\tau) < d(\hat{s}(i), x)$. Also, since d is a metric : $d(\hat{s}(i), x) \leq d(\hat{s}(i), s(t)) + d(s(t), x)$. Hence, $d(s(t), x) > 0$. Since this holds for any $x \in X \backslash \mathcal{O}(p)$, we conclude that $\mathbf{depth}_d(s(t), \mathcal{O}(p)) > 0$ or $s(t) \in \mathcal{O}(p)$ and, thus, $\langle\!\langle p \rangle\!\rangle(s,t) = \top$ for all $t \in [\tau(i) - \Delta\tau, \tau(i) + \Delta\tau] \cap R$.

Case $\phi = \neg p \in AP$: $[\![\mathcal{H}(\neg p)]\!]_\tau(s,i) > \mathcal{E}(\Delta\tau)$, i.e., $\mathbf{dist}_d(\hat{s}(i), \mathcal{O}(p)) > \mathcal{E}(\Delta\tau)$. The proof is similar to the previous case.

Cases $\phi = \phi_1 \vee \phi_2$ and $\phi = \phi_1 \wedge \phi_2$: straightforward.

Case $\phi = \phi_1 \mathcal{U}_\mathcal{I} \phi_2$: We know that $[\![\mathcal{H}(\phi_1) \mathcal{U}_{C(\mathcal{I},\Delta\tau)} \mathcal{H}(\phi_2)]\!]_\tau(s,i) > \mathcal{E}(\Delta\tau)$. By Lemma 1 and the definition of until : there exists a $j \in \tau^{-1}(\tau(i) +_R C(\mathcal{I}, \Delta\tau))$ such that $[\![\mathcal{H}(\phi_2)]\!]_\tau(s,j) > \mathcal{E}(\Delta\tau)$ and for all k such that $i \leq k < j$ we have $[\![\mathcal{H}(\phi_1)]\!]_\tau(s,k) > \mathcal{E}(\Delta\tau)$. By the induction hypothesis, we get that $\langle\!\langle \phi_2 \rangle\!\rangle(s,t) = \top$ for all $t \in [\tau(j) - \Delta\tau, \tau(j) + \Delta\tau] \cap R$ and $\langle\!\langle \phi_1 \rangle\!\rangle(s,t) = \top$ for all $t \in [\tau(k) - \Delta\tau, \tau(k) + \Delta\tau] \cap R$ and for all $k \in [i,j)$. We set $t' = \tau(j)$. Note that for all $t \in [\tau(i) - \Delta\tau, \tau(i) + \Delta\tau] \cap R$ we have $t' \in t +_R \mathcal{I}$ since $\tau(j) \in \tau(i) +_R C(\mathcal{I}, \Delta\tau)$. Also, since $\tau(j) \leq \tau(j-1) + \Delta\tau$ we get that for all $t'' \in [t, t')$ we have $\langle\!\langle \phi_1 \rangle\!\rangle(s,t'') = \top$. Hence, we conclude that $\langle\!\langle \phi_1 \mathcal{U}_\mathcal{I} \phi_2 \rangle\!\rangle(s,t) = \top$ for all $t \in [\tau(i) - \Delta\tau, \tau(i) + \Delta\tau] \cap R$.

Case $\phi = \phi_1 \mathcal{R}_{\mathcal{I}} \phi_2$: We know that $[\![\mathcal{H}(\phi_1)\mathcal{R}_{E(\mathcal{I},\Delta\tau)}\mathcal{H}(\phi_2)]\!]_\tau(s,i) > \mathcal{E}(\Delta\tau)$. By Lemma 1 and the definition of release : for all $j \in \tau^{-1}(\tau(i) +_R E(\mathcal{I},\Delta\tau))$ we have $[\![\mathcal{H}(\phi_2)]\!]_\tau(s,j) > \mathcal{E}(\Delta\tau)$ or there exists k such that $i \leq k < j$ and $[\![\mathcal{H}(\phi_1)]\!]_\tau(s,k) > \mathcal{E}(\Delta\tau)$. By the induction hypothesis, we get that for all $j \in \tau^{-1}(\tau(i) +_R E(\mathcal{I},\Delta\tau))$ we have $\langle\!\langle \phi_2 \rangle\!\rangle(s,t) = \top$ for all $t \in [\tau(j) - \Delta\tau, \tau(j) + \Delta\tau] \cap R$ and $\langle\!\langle \phi_1 \rangle\!\rangle(s,t) = \top$ for all $t \in [\tau(k) - \Delta\tau, \tau(k) + \Delta\tau] \cap R$. Let $j_m = \min \tau^{-1}(\tau(i) +_R E(\mathcal{I},\Delta\tau))$ and $j_M = \max \tau^{-1}(\tau(i) +_R E(\mathcal{I},\Delta\tau))$. For all $t' \in [\tau(j_m) - \Delta\tau, \tau(j_M) + \Delta\tau]$ we have $\langle\!\langle \phi_2 \rangle\!\rangle(s,t') = \top$. But, for all $t \in [\tau(i) - \Delta\tau, \tau(i) + \Delta\tau] \cap R$ we have $t +_R \mathcal{I} \subseteq \tau(i) +_R E(\mathcal{I},\Delta\tau)$. Hence, for all $t \in [\tau(i) - \Delta\tau, \tau(i) + \Delta\tau] \cap R$, for all $t' \in t +_R \mathcal{I}$, we have $\langle\!\langle \phi_2 \rangle\!\rangle(s,t') = \top$ or there exists some $t'' \in [t,t')$ such that $\langle\!\langle \phi_1 \rangle\!\rangle(s,t'') = \top$. Hence, $\langle\!\langle \phi_1 \mathcal{R}_{\mathcal{I}} \phi_2 \rangle\!\rangle(s,t) = \top$ for all $t \in [\tau(i) - \Delta\tau, \tau(i) + \Delta\tau] \cap R$. □

We should remark that the conclusion (7) of Theorem 1 does not imply that the continuous time Boolean signal $\mathcal{O}^{-1} \circ s$ satisfies the finite variability property as it is defined in [5]. It only states that there exists some time interval in R of length at least $2\Delta\tau$ such that the Boolean truth value of some atomic propositions remains constant.

Corollary 1. *Consider* $\phi \in \Phi_{\mathbb{B}}$, $\mathcal{O} \in \mathcal{P}(X)^{AP}$, $s \in \mathfrak{S}$, $\tau \in \mathfrak{T}$ *and let Assumptions 1-3 hold. Then,* $[\![\mathcal{H}(\phi)]\!]_\tau(s) > \mathcal{E}(\Delta\tau)$ *implies* $\langle\!\langle \phi \rangle\!\rangle(s) = \top$.

If the condition $[\![\mathcal{H}(\phi)]\!]_\tau(s) > \mathcal{E}(\Delta\tau)$ fails, then in general we cannot infer anything about the relationship of the two semantics. Two strategies in order to guarantee the above condition would be (i) to reduce the size of the sampling step $\Delta\tau$ or (ii) to devise an on-line monitoring procedure that can adjust real-time the sampling step according to the robustness estimate of a signal with respect to an MITL formula ϕ.

6 Examples

In this section, we demonstrate the proposed methodology with some examples. The discrete time signals under consideration could be the result of sampling a physical signal or a simulated one. The latter is meaningful in cases where we would like to use fewer sampled points for temporal logic testing, while simulating the actual trajectory with finer integration step. The robustness estimate is computed using the algorithm that was presented in [12].

Example 2. *Assume that we are given a discrete representation of a signal* $\hat{s}_1$ *(Fig. 1) which has constant sampling step of magnitude* 0.2*, i.e.,* $\Delta\tau_1 = 0.2$*. We are also provided with the constraint* $\mathcal{E}_1(t) = 3t$ *(notice that* $|\dot{s}_1(t)| \leq |\cos t| + 2|\cos 2t| \leq 1 + 2 = 3$ *for all* $t \in R$*, therefore* s_1 *is Lipschitz continuous with* $L_{s_1} = 3$*). We would like to test whether the underlying continuous time signal* s_1 *satisfies the specification* $\phi_1 = \Box_{[0,9\pi/2]}(p_{11} \rightarrow \Diamond_{[\pi,2\pi]} p_{12})$*, with* $\mathcal{O}(p_{11}) = \mathbb{R}_{\geq 1.5}$ *and* $\mathcal{O}(p_{12}) = \mathbb{R}_{\leq -1}$*. Notice that the sampling function* τ_1 *satisfies the constraints of the Assumptions 2 and 3. Using the computational procedure proposed in [12], we compute a robustness estimate of* $[\![\mathcal{H}(\phi_1)]\!]_{\tau_1}(s_1) = 0.7428$*, while* $\mathcal{E}_1(\Delta\tau_1) = 0.6$*. Therefore, by Corollary 1 we conclude that* $\langle\!\langle \phi_1 \rangle\!\rangle(s_1) = \top$. □

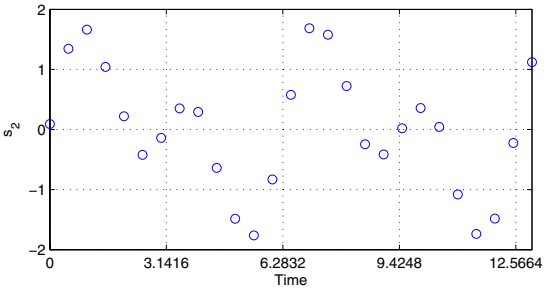


Fig. 2. The sampled signal $\hat{s}_2$ generated by sampling the continuous time signal $s_2(t) = \sin(t) + \sin(2t) + w(t)$, where $|w(t)| \leq 0.1$, with constant sampling period 0.5. In this case, it is $|s_2(t_1) - s_2(t_2)| \leq L_{s_1}|t_1 - t_2| + |w(t_1)| + |w(t_2)|$. Thus, $\mathcal{E}_2(t) = L_{s_1}t + 0.2$.

The next example manifests a very intuitive attribute of the framework, namely, that the more robust a signal is with respect to the MITL specification the larger the sampling period can be.

Example 3. *Consider the discrete time signal $\hat{s}_2$ in Fig. 2. The MITL specification is $\phi_2 = \Box_{[0,4\pi]}p_{21} \wedge \Diamond_{[3\pi,4\pi]}p_{22}$ with $\mathcal{O}(p_{21}) = [-4, 4]$ and $\mathcal{O}(p_{22}) = \mathbb{R}_{\leq 0}$. In this case, we compute a robustness estimate of $[\![\mathcal{H}(\phi_2)]\!]_{\tau_2}(s_2) = 1.7372$, while $\mathcal{E}_2(\Delta\tau_2) = 1.7$ where $\Delta\tau_2 = 0.5$. Therefore, we conclude that $\langle\!\langle \phi_2 \rangle\!\rangle(s_2) = \top$.* □

In the following example, we utilize our framework in order to test trajectories of nonlinear systems. More specifically, we consider linear feedback systems with saturation. Such systems have nonlinearities that model sensor/actuator constraints (for example see [8, §10]).

Example 4 (Example 10.5 in [8]). *Consider the following linear dynamical system with nonlinear feedback*

$$\dot{x}(t) = Ax(t) - b\,\mathrm{sat}(cx(t)), \quad s_3(t) = cx(t) \tag{8}$$

where the saturation function sat *is defined as*

$$\mathrm{sat}(y) = \begin{cases} -1 & \text{for } y < -1 \\ y & \text{for } |y| \leq 1 \\ 1 & \text{for } y > 1 \end{cases}$$

and A, b, c are the matrices

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad c = \begin{bmatrix} 2 & 1 \end{bmatrix}.$$

First note that the origin $x = [0\ 0]^T$ is an equilibrium point of the system and that the system is absolutely stable with a finite domain (also note that A is

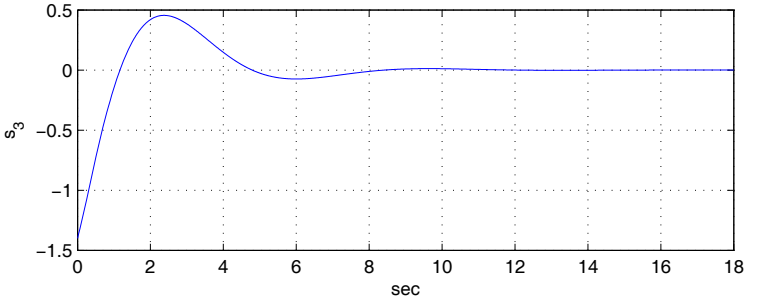


Fig. 3. The output signal s_3 of Example 4

not Hurwitz). An estimate of the region of attraction of the origin is the set $\Omega = \{x \in \mathbb{R}^2 \mid V(x) \leq 0.34\}$, *where* $V(x) = x^T P x$ *and*

$$P = \begin{bmatrix} 0.4946 & 0.4834 \\ 0.4834 & 1.0774 \end{bmatrix}$$

(see Example 10.5 in [8] for details). For any initial condition $x(0) \in \Omega$, *we know that* $x(t) \in \{x \in \mathbb{R}^2 \mid V(x) \leq V(x(0))\}$ *for all* $t \in R$. *Thus,* $\|x(t)\| \leq \sqrt{\lambda_{\max}(V(x(0))P^{-1})} = \sqrt{\lambda_{\max}(P_e)}$ *for all* $t \in R$. *Moreover,*

$$\|\dot{x}(t)\| \leq \|A\|\|x(t)\| + \|b\| \leq \|A\|\sqrt{\lambda_{\max}(P_e)} + \|b\| = L_x$$

and, thus, we have $|s_3(t) - s_3(t')| \leq \|c\|\|x(t) - x(t')\| \leq \|c\|L_x|t - t'|$ *for any* $t, t' \in R$, *i.e.,* $\mathcal{E}_3(t) = \|c\|L_x t$. *Assume, now, that we would like to verify that the signal enters an acceptable stability region within* 6 *to* 8 sec, *that is, the MITL formula is* $\phi_3 = \Diamond_{[6,8]}\Box_{[0,10]} p_{31}$ *with* $\mathcal{O}(p_{31}) = [-0.25, 0.25]$. *The initial condition is* $x(0) = [-1\ 0.6]^T \in \Omega$. *The system (8) is integrated with a maximum step-size of* 0.001 *using the MATLAB ode45 solver. The observable discrete time signal* $\hat{s}_3$ *has maximum step-size* $\Delta\tau_3 = 0.045$. *The robustness estimate is* $[\![\mathcal{H}(\phi_3)]\!]_{\tau_3}(s_3) = 0.2372$, *while* $\mathcal{E}_3(\Delta\tau_3) = 0.2182$. *Therefore, we conclude that* $\langle\!\langle \phi_3 \rangle\!\rangle(s_3) = \top$. *Note that in this example, we assume that the simulation is accurate and, hence, we ignore the possible simulation error. The incorporation of the simulation error into* $\mathcal{E}_3$ *will be part of future research.* □

7 Conclusions and Discussion

We have developed a framework that enables continuous time reasoning using discrete time methods. The target application is on continuous time signals generated by physical systems with real-time constraints. Our solution utilizes the notion of robustness of MTL specifications [12] and provides conditions on the signal dynamics and the sampling function.

We should point out that the idea of continuous time verification by discrete reasoning is not new. In [20], the authors show that if a formula has the finite

variability property, then its validity in discrete time implies validity in continuous time. This result enables the application of verification rules for discrete time semantics to continuous time problems. The work that is the most related to ours appears in [21]. There, the authors give conditions that enable the uniform treatment of both discrete and continuous time semantics within the temporal logic TRIO (they also note that their results should be easily transferable to MTL). Despite the apparent differences (for example, we do not assume finite variability and we use analog clocks in our discrete time logic) between [21] and our work, the two papers are in fact complementary. We actually provide concrete and practical conditions on the signals such that what is defined as "closure under inverse sampling" in [21] holds.

In the current framework, we require a global bound $\mathcal{E}(\Delta\tau)$ on the deviation of the signal between two samples. This might be too conservative for applications with variable sampling step. One important modification to this theory will be to use local bounds $\mathcal{E}(\tau(i)-\tau(i-1))$ in coordination with an on-line monitoring algorithm. Related to the previous modification is the extension of the present methodology to hybrid systems [22]. Currently, hybrid systems can be handled by taking as bound $\mathcal{E}$ the most conservative bound $\mathcal{E}_c$ of all control locations c of the hybrid automaton. Finally, as it is well known, the Lipschitz constant might be a very conservative estimate on the deviation of the signal between two points in time. In future work, we plan to use approximate metrics [23] in order to obtain better bounds.

Acknowledgments. The authors would like to thank A. Agung Julius for the useful discussions. This work has been partially supported by NSF EHS 0311123, NSF ITR 0324977 and ARO MURI DAAD 19-02-01-0383.

References

1. Pillage, L., Rohrer, R., Visweswariah, C.: Electronic Circuit and System Simulation Methods. McGraw-Hill, New York (1995)
2. Ogata, K.: Modern Control Engineering, 4th edn. Prentice-Hall, Englewood Cliffs (2001)
3. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems 2, 255–299 (1990)
4. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM 43, 116–146 (1996)
5. Hirshfeld, Y., Rabinovich, A.: Logics for real time: Decidability and complexity. Fundam. Inf. 62, 1–28 (2004)
6. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004)
7. Maler, O., Nickovic, D., Pnueli, A.: From MITL to Timed Automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 274–289. Springer, Heidelberg (2006)
8. Khalil, H.K.: Nonlinear Systems, 2nd edn. Prentice-Hall, Englewood Cliffs (1996)

9. Thati, P., Rosu, G.: Monitoring algorithms for metric temporal logic specifications. In: Runtime Verification. ENTCS, vol. 113, pp. 145–162. Elsevier, Amsterdam (2005)
10. Bensalem, S., Bozga, M., Krichen, M., Tripakis, S.: Testing conformance of real-time applications with automatic generation of observers. In: Proceedings of Runtime Verification Workshop, Barcelona, Spain (2004)
11. Tan, L., Kim, J., Sokolsky, O., Lee, I.: Model-based testing and monitoring for hybrid embedded systems. In: Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, pp. 487–492 (2004)
12. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES 2006 and RV 2006. LNCS, vol. 4262, pp. 178–192. Springer, Heidelberg (2006)
13. Pnueli, A.: Development of hybrid systems. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS, vol. 863, pp. 77–85. Springer, Heidelberg (1994)
14. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
15. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science: Formal Models and Semantics, vol. B, pp. 995–1072. North-Holland Pub. Co./MIT Press, Amsterdam (1990)
16. Alur, R., Henzinger, T.A.: Real-Time Logics: Complexity and Expressiveness. In: Fifth Annual IEEE Symposium on Logic in Computer Science, pp. 390–401. IEEE Computer Society Press, Washington, D.C (1990)
17. de Alfaro, L., Faella, M., Stoelinga, M.: Linear and branching metrics for quantitative transition systems. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 97–109. Springer, Heidelberg (2004)
18. Wood, G.R., Zhang, B.P.: Estimation of the Lipschitz constant of a function. Journal of Global Optimization 8, 91–103 (1996)
19. Huang, J., Voeten, J., Geilen, M.: Real-time property preservation in approximations of timed systems. In: Proceedings of the 1st ACM & IEEE International Conference on Formal Methods and Models for Co-Design, pp. 163–171 (2003)
20. de Alfaro, L., Manna, Z.: Verification in continuous time by discrete reasoning. In: Alagar, V.S., Nivat, M. (eds.) AMAST 1995. LNCS, vol. 936, pp. 292–306. Springer, Heidelberg (1995)
21. Furia, C.A., Rossi, M.: Integrating discrete and continuous time metric temporal logics through sampling. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 215–229. Springer, Heidelberg (2006)
22. Julius, A.A., Fainekos, G.E., Anand, M., Lee, I., Pappas, G.J.: Robust test generation and coverage for hybrid systems. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 329–342. Springer, Heidelberg (2007)
23. Girard, A., Pappas, G.J.: Approximation metrics for discrete and continuous systems. IEEE Trans. Auto. Cont. 52, 782–798 (2007)

On the Expressiveness of MTL Variants over Dense Time

Carlo A. Furia and Matteo Rossi

Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
{furia,rossi}@elet.polimi.it
http://home.dei.polimi.it/lastname/

Abstract. The basic modal operator *bounded until* of Metric Temporal Logic (MTL) comes in several variants. In particular it can be *strict* (when it does not constrain the current instant) or not, and *matching* (when it requires its two arguments to eventually hold together) or not. This paper compares the relative expressiveness of the resulting MTL variants over dense time. We prove that the expressiveness is not affected by the variations when considering non-Zeno interpretations and arbitrary nesting of temporal operators. On the contrary, the expressiveness changes for flat (i.e., without nesting) formulas, or when Zeno interpretations are allowed.

1 Introduction

In the last few decades, the formal description and analysis of real-time systems has become an increasingly important research topic. This has resulted, among other things, in the development of several formal notations for the description of real-time properties and systems. In particular, a significant number of extensions of classical temporal logics to deal with metric (quantitative) time has been introduced and used (see e.g., [3]). Among them, Metric Temporal Logic (MTL) [23,4] is one of the most popular. An appealing feature of MTL is its being a straightforward extension of well-known Linear Temporal Logic (LTL), a classical temporal logic. In MTL, an interval parameter is added to LTL's modal operators (such as the *until* operator); the interval specifies a range of distances over which the arguments of the modality must hold, thus allowing the expression of real-time properties.

When MTL formulas are interpreted over discrete time domains (e.g., $\mathbb{N}, \mathbb{Z}$), the well-known results and techniques about the expressiveness of LTL can often be "lifted" to the real-time case [4]. On the contrary, when MTL formulas are interpreted over dense time domains (e.g., $\mathbb{R}$) additional difficulties and complications are commonly encountered, which require novel techniques (e.g., [20,21,5,11,29,28,6]). Another aspect where the use of MTL (and temporal logics in general) over metric dense-time models shows a substantial difference with respect to discrete time is in the *robustness* of the language expressiveness with respect to changes in its (syntactic) definitions or in the choice of the underlying interpretation structures. In other words, it is often the case that apparently

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 163–178, 2007.

minimal changes in the definition of the basic modal operators, or in the choice of the interpretation structures (e.g., timed words rather than timed interval sequences), of the logics yield substantial differences in the resulting expressiveness. Also, these differences are usually more difficult to predict and assess than in the discrete-time case. One significant example is the use of a "natural" extension such as the introduction of past operators: it is well-known that adding them does not change the expressive power over discrete time [12] (while it increases the succinctness [25]), but it does over dense time both for LTL [18,22], and for MTL [5,29] (Alur and Henzinger [2] were the first to analyze this issue for a metric MTL subset known as MITL).

This paper contributes to enriching the emerging picture about the expressiveness of MTL and its common variants. The reference interpretation structure is the *behavior*, that is generic mappings that associate with every instant of time the propositions that are true at that instant. When behaviors are restricted to be non-Zeno [19] (also called *finitely variable* [30,20]) they are an equivalent way of expressing the well-known timed interval sequences. We consider two basic language features to be varied in MTL definitions: strictness and matchingness. The basic *until* operator $\mathsf{U}(\phi_1, \phi_2)$ is called *strict* (in its first argument) if it does not constrain its first argument ϕ_1 to hold at the current instant (i.e., it constraints strictly the future); on the other hand the same operator is called *matching* if it requires the second argument ϕ_2 to hold *together with* the first argument ϕ_1 at some instant in the future (see Section 2 for precise definitions). The most common definition uses an *until* operator that is strict and non-matching; it is simple to realize that this does not restrict the expressiveness as the matching and non-strict *until*s are easily expressible in terms of strict non-matching *until*s. However, some applications dealing with MTL or closely related languages are based on the matching (e.g., [26,27]) or non-strict (e.g., [15]) variants, or both. Therefore it is interesting to analyze if these syntactic restrictions imply restrictions in the expressiveness of the language; this is done in Section 3.

Another dimension that we consider in our analysis is the restriction to *flat* MTL formulas, i.e., formulas that do not nest temporal operators. These have also been used, among others, in some previous work of ours [15], as well as in several works with classical (qualitative) temporal logic (see related works). It is an easy guess that flat MTL is less expressive than its full "nesting" counterpart; in this paper (Section 4) we prove this intuition and then we analyze how the relationships between the various (non-)matching and (non-)strict variants change when restricted to flat formulas. In order to do so, we develop techniques to handle two different definitions in the satisfaction semantics of formulas: initial satisfiability — where the truth of a formula is evaluated only at some initial time (i.e., 0) — and global satisfiability — where the truth of a formula is evaluated at all time instants. While the two semantics are easily reconcilable when nesting is allowed, passing from the initial semantics to the global one with flat formulas is more challenging. We consider also the less common global satisfiability semantics because the expressiveness of the flat fragment is non-trivial under such semantics (in fact, it corresponds to an implicit nesting of a

qualitative temporal operator), and most common real-time properties such as bounded response and bounded invariance [23] can be easily expressed.

Finally, in Section 5, we also consider what happens to (relative) expressiveness if Zeno behaviors are allowed. In particular, we show that some equivalences between MTL variations that hold over non-Zeno behaviors are no more valid with Zeno behaviors. In this sense, allowing Zeno behaviors weakens the robustness of MTL expressiveness with respect to definitions, and it renders the picture more complicated and less intuitive.

For lack of space, we have omitted several proofs and details from this version of the paper; the interested reader can find them in [17].

Related works. As mentioned above, in recent years several works have analyzed the expressiveness of different MTL variants over dense time. Let us recall briefly the main results of a significant subset thereof.

Bouyer et al. [5] compare the expressiveness of MTL with that of TPTL (another real-time temporal logic) and they are the first to prove the conjecture that the latter is strictly more expressive than the former, over both timed words and timed interval sequences. As a corollary of their results, they show that past operators increase the expressiveness of MTL.

Since the work of Alur and Henzinger [4] it is known that (full) MTL is undecidable over dense-time models. This shortcoming has been long attributed to the possibility of expressing punctual (i.e., exact) timing constraints; in fact Alur et al. [1] have shown that MITL, a MTL subset where punctual intervals are disallowed, is decidable. However, punctuality does not always entail undecidability. In fact, Ouaknine and Worrel [28] have been the first to prove that MTL is decidable over finite timed words, albeit with non-primitive recursive complexity; their proofs rely on automata-based techniques, and in particular on the notion of timed automata with alternation. On the other hand, they show that several significant fragments of MTL are still undecidable over infinite timed words. In the same vein, Bouyer et al. [6] have identified significant MTL fragments that are instead decidable (with primitive complexity) even if one allows the expression of punctual timing constraints.

D'Souza and Prabhakar [11] compare the expressiveness of MTL over the two interpretation structures of timed words and timed interval sequences (more precisely, a specialization of the latter called "continuous" semantics). Building upon Ouaknine and Worrel's decidability results for MTL [28], they show that MTL is strictly more expressive over timed interval sequences than it is over timed words. The same authors [29] analyze a significant number of MTL variations, namely those obtained by adding past operators or by considering qualitative operators rather than metric ones, over both timed words and timed interval sequences, both in their finite and infinite forms. Still the same authors [10] have shown how to rewrite MTL formulas in flat form, and without past operators, by introducing additional propositions (a similar flattening has been shown for another temporal logic in [14]). While these latter results do not pertain directly to the expressiveness of the language (because of the new propositions that are introduced) they help assessing the decidability of MTL variations.

In previous work [16] we proved the equivalence between the strict and non-strict non-matching variants of MTL over non-Zeno behaviors and with arbitrarily nested formulas. Section 3 uses techniques similar to those in [16] to prove new equivalence results.

These results, which we do not report with further details for the lack of space, show how relative expressiveness relations are much more complicated over dense time than they are over discrete time. In fact, some authors (e.g., [20,3]) have suggested that these additional difficulties are an indication that the "right" semantic model for dense time has not been found yet. In particular, Hirshfeld and Rabinovich [20,21] have made a strong point that most approaches to the definition of temporal logics for real-time and to their semantics depart from the "classical" approach to temporal logic and are too *ad hoc*, which results in unnecessary complexity and lack of robustness. While we agree with several of their remarks, we must also acknowledge that MTL (and other similar logics) has become a popular notation, and it has been used in several works. As a consequence, it is important to assess precisely the expressiveness of the language and of its common variants because of the impact on the scope of those works, even if focusing on different languages might have opened the door to more straightforward approaches.

Finally, let us mention that several works dealing with classical (qualitative) temporal logic considered variants in the definition of the basic modalities, and their impact on expressiveness and complexity. For instance, Demri and Schnoebelen [9] thoroughly investigate the complexity of LTL without nesting, or with a bounded nesting depth. Also, several works have given a very detailed characterization of how the expressiveness of LTL varies with the number of nested modalities [13,32,24]; and several other works, such as [8,7], have used and characterized flat fragments where nesting is only allowed in the second argument of any *until* formula. Reynolds [31] has proved that, over dense time, LTL with strict until is strictly more expressive than LTL with a variant of non-strict until which includes the current instant. Note that Reynold's non-strict until has a different (weaker) semantics than the one we consider in this paper, because of the restriction to include the current instant. In other words, according to the notation that we introduce in Section 2, [31] compares the strict $\widetilde{\mathsf{U}}_{(0,+\infty)}$ to the non-strict $\mathsf{U}_{[0,+\infty)}$; as a consequence, Reynold's result is orthogonal to ours.

2 MTL and Its Variants

MTL is built out of the single modal operator bounded *until*[1] through propositional composition. Formulas are built according to the grammar: $\phi ::= \mathsf{p} \mid \widetilde{\mathsf{U}}_I(\phi_1, \phi_2) \mid \neg\phi \mid \phi_1 \wedge \phi_2$ where I is an interval $\langle l, u\rangle$ of the reals such that $0 \le l \le u \le +\infty$, $l \in \mathbb{Q}$, $u \in \mathbb{Q} \cup \{+\infty\}$, and $\mathsf{p} \in \mathcal{P}$ is some atomic proposition from a finite set $\mathcal{P}$.

The tilde in $\widetilde{\mathsf{U}}_I$ denotes that the *until* is *strict*, as it will be apparent in the definition of its semantics; $\widetilde{\mathsf{U}}_I$ is also meant to be *non-matching*. We denote the

[1] In this paper we consider MTL with future operators only.

set of formulas generated by the grammar above as $\widetilde{\text{MTL}}$, which is therefore strict non-matching.

We define formally the semantics of $\widetilde{\text{MTL}}$ over generic Boolean behaviors. Given a time domain $\mathbb{T}$ and a finite set of atomic propositions $\mathcal{P}$, a *Boolean behavior* over $\mathcal{P}$ is a mapping $b : \mathbb{T} \to 2^{\mathcal{P}}$ from the time domain to subsets of $\mathcal{P}$: for every time instant $t \in \mathbb{T}$, b maps t to the set of propositions $b(t)$ that are true at t. We denote the set of all mappings for a given set $\mathcal{P}$ as $\mathcal{B}_{\mathcal{P}}$, or simply as $\mathcal{B}$. In practice, in this paper we take $\mathbb{T}$ to be the reals $\mathbb{R}$, but all our results hold also for $\mathbb{R}_{\geq 0}, \mathbb{Q}, \mathbb{Q}_{\geq 0}$ as time domains.[2]

The semantics of $\widetilde{\text{MTL}}$ formulas is given through a satisfaction relation $\models_{\mathbb{T}}$: given a behavior $b \in \mathcal{B}$, an instant $t \in \mathbb{T}$ (sometimes called "current instant") and an $\widetilde{\text{MTL}}$ formula ϕ, the satisfaction relation is defined inductively as follows.

$b(t) \models_{\mathbb{T}} \mathsf{p}$ iff $\mathsf{p} \in b(t)$

$b(t) \models_{\mathbb{T}} \widetilde{\mathsf{U}}_I(\phi_1, \phi_2)$ iff there exists $d \in I$ such that $b(t+d) \models_{\mathbb{T}} \phi_2$ and, for all $u \in (0, d)$ it is $b(t+u) \models_{\mathbb{T}} \phi_1$

$b(t) \models_{\mathbb{T}} \neg\phi$ iff $b(t) \not\models_{\mathbb{T}} \phi$

$b(t) \models_{\mathbb{T}} \phi_1 \wedge \phi_2$ iff $b(t) \models_{\mathbb{T}} \phi_1$ and $b(t) \models_{\mathbb{T}} \phi_2$

From these definitions, we introduce initial satisfiability and global satisfiability as follows: a formula ϕ is *initially satisfiable* over a behavior b iff $b(0) \models_{\mathbb{T}} \phi$; a formula ϕ is *globally satisfiable* over a behavior b iff $\forall t \in \mathbb{T} : b(t) \models_{\mathbb{T}} \phi$, and we write $b \models_{\mathbb{T}} \phi$. The initial and global satisfiability relations allow one to identify a formula ϕ with the set of behaviors $[\![\phi]\!]_{\mathbb{T}}$ that satisfy it according to each semantics; hence we introduce the notation $[\![\phi]\!]^0_{\mathbb{T}} = \{b \in \mathcal{B} \mid b(0) \models_{\mathbb{T}} \phi\}$ and $[\![\phi]\!]_{\mathbb{T}} = \{b \in \mathcal{B} \mid b \models_{\mathbb{T}} \phi\}$.

From the basic strict operator we define syntactically some *variants*: the non-strict non-matching *until* U_I, the strict matching *until* $\widetilde{\mathsf{U}}^{\downarrow}_I$, and the non-strict matching *until* $\mathsf{U}^{\downarrow}_I$; they are defined in Table 1. Also, we define the following derived modal operators:[3] $\widetilde{\mathsf{R}}^{\downarrow}_I(\phi_1, \phi_2) \equiv \neg\widetilde{\mathsf{U}}^{\downarrow}_I(\neg\phi_1, \neg\phi_2)$, $\widetilde{\Diamond}_I(\phi) \equiv \widetilde{\mathsf{U}}_I(\top, \phi)$, $\widetilde{\Box}_I(\phi) \equiv \neg\widetilde{\Diamond}_I(\neg\phi)$, $\bigcirc(\phi) \equiv \mathsf{U}_{(0,+\infty)}(\phi, \top)$, and $\widetilde{\bigcirc}(\phi) \equiv \widetilde{\mathsf{U}}_{(0,+\infty)}(\phi, \top)$; derived propositional connectives (such as $\Rightarrow, \vee, \Leftrightarrow$) are defined as usual. For derived operators we use the same notational conventions: a $\sim$ denotes strictness and a $\downarrow$ denotes matchingness. Accordingly, we denote by MTL the set of non-strict non-matching formulas (i.e., those using only the U_I operator), by $\widetilde{\text{MTL}}^{\downarrow}$ the set of strict matching formulas (i.e., those using only the $\widetilde{\mathsf{U}}^{\downarrow}_I$ operator), and by $\text{MTL}^{\downarrow}$ the set of non-strict matching formulas (i.e., those using only the $\mathsf{U}^{\downarrow}_I$ operator).

Note that the $\widetilde{\Diamond}$ operator (and correspondingly the $\widetilde{\Box}$ operator as well) can be equivalently expressed with any of the *until* variants introduced beforehand, i.e., $\widetilde{\Diamond}_I(\phi) \equiv \widetilde{\mathsf{U}}_I(\top, \phi) \equiv \mathsf{U}_I(\top, \phi) \equiv \widetilde{\mathsf{U}}^{\downarrow}_I(\top, \phi) \equiv \mathsf{U}^{\downarrow}_I(\top, \phi)$. Therefore, in the following we drop the tilde and write $\Diamond_I$ (resp. $\Box_I$) in place of $\widetilde{\Diamond}_I$ (resp. $\widetilde{\Box}_I$).

[2] Even if we deal only with future operators, bi-infinite time domains $\mathbb{R}$ and $\mathbb{Q}$ are considered as they match "more naturally" the global satisfiability semantics.

[3] For clarity, let us give explicitly the semantics of the $\widetilde{\mathsf{R}}^{\downarrow}_I$ operator: $b(t) \models_{\mathbb{T}} \widetilde{\mathsf{R}}^{\downarrow}_I(\phi_1, \phi_2)$ iff for all $d \in I$ it is: $b(t+d) \models_{\mathbb{T}} \phi_2$ or $b(t+u) \models_{\mathbb{R}} \phi_1$ for some $u \in (0, d]$.

Table 1. *Until* operator variants

Operator	$\equiv$	Definition
$\mathsf{U}_I(\phi_1, \phi_2)$	$\equiv$	if $0 \notin I$: $\phi_1 \wedge \widetilde{\mathsf{U}}_I(\phi_1, \phi_2)$ else: $\phi_2 \vee (\phi_1 \wedge \widetilde{\mathsf{U}}_I(\phi_1, \phi_2))$
$\widetilde{\mathsf{U}}^{\downarrow}_I(\phi_1, \phi_2)$	$\equiv$	if $0 \notin I$: $\widetilde{\mathsf{U}}_I(\phi_1, \phi_2 \wedge \phi_1)$ else: $\phi_2 \vee (\widetilde{\mathsf{U}}_I(\phi_1, \phi_2 \wedge \phi_1))$
$\mathsf{U}^{\downarrow}_I(\phi_1, \phi_2)$	$\equiv$	$\phi_1 \wedge \widetilde{\mathsf{U}}_I(\phi_1, \phi_2 \wedge \phi_1) \equiv \mathsf{U}_I(\phi_1, \phi_2 \wedge \phi_1)$

According to the semantics, all formulas of some MTL variant identify a set of sets of behaviors which characterize the *expressive power* of that variant. We overload the notation and also denote by $\widetilde{\text{MTL}}$, MTL, $\widetilde{\text{MTL}}^{\downarrow}$, and $\text{MTL}^{\downarrow}$ the set of sets of behaviors identified by all strict non-matching, non-strict matching, strict matching, and non-strict non-matching formulas, respectively. It will be clear from the context whether we are referring to a set of formulas or to the corresponding set of sets of behaviors, and whether we are considering the initial or global satisfiability semantics.

For every formula ϕ, we define its *granularity* ρ as the reciprocal of the product of all denominators of non-null finite interval bounds appearing in ϕ; and its *nesting depth* (also called *temporal height*) k as the maximum number of nested modalities in ϕ. A formula is called *flat* if it does not nest modal operators, and *nesting* otherwise. Given a set of formulas F, the subset of all its flat formulas is denoted by $\flat F$ (for instance flat non-strict non-matching formulas are denoted as $\flat$MTL).

Since the non-strict and matching variants have been defined in terms of $\widetilde{\text{MTL}}$ — and their definitions do not nest temporal operators — it is clear that the following relations hold: $\text{MTL}^{\downarrow} \subseteq \text{MTL} \subseteq \widetilde{\text{MTL}}$, $\text{MTL}^{\downarrow} \subseteq \widetilde{\text{MTL}}^{\downarrow} \subseteq \widetilde{\text{MTL}}$, $\flat\text{MTL}^{\downarrow} \subseteq \flat\text{MTL} \subseteq \flat\widetilde{\text{MTL}}$, and $\flat\text{MTL}^{\downarrow} \subseteq \flat\widetilde{\text{MTL}}^{\downarrow} \subseteq \flat\widetilde{\text{MTL}}$.

Non-Zenoness. Behaviors over dense time are often subject to the *non-Zenoness* (also called *finite variability* [20,30]) requirement [19]. A behavior $b \in \mathcal{B}$ is called *non-Zeno* if the truth value of any atomic proposition $\mathsf{p} \in \mathcal{P}$ changes in b only finitely many times over any bounded interval of time. In [16] we proved that strict $\widetilde{\bigcirc}$ operator can be expressed with non-strict $\bigcirc$ operator over non-Zeno behaviors as $\widetilde{\bigcirc}(\phi) \equiv \bigcirc(\phi) \vee (\neg\phi \wedge \neg \bigcirc(\neg\phi))$.

3 Nesting MTL over Non-zeno Behaviors

This section shows that the four MTL variants: $\widetilde{\text{MTL}}$, MTL, $\widetilde{\text{MTL}}^{\downarrow}$, and $\text{MTL}^{\downarrow}$ all have the same expressive power over non-Zeno behaviors, for both the initial and global satisfiability semantics. In fact, we provide a set of equivalences according to which one can replace each occurrence of strict *until* in terms of non-strict *until*, and each occurrence of non-matching *until* in terms of matching *until*. This shows that $\text{MTL} = \widetilde{\text{MTL}} = \widetilde{\text{MTL}}^{\downarrow} = \text{MTL}^{\downarrow}$. Note that the result holds regardless of whether the global or initial satisfiability relation in considered.

3.1 Non-strict as Expressive as Strict

In [16] we have shown that MTL = $\widetilde{\text{MTL}}$; more precisely, the following equivalences have been proved, for $a > 0$ (and $b > 0$ in (4)).

$$\widetilde{\mathsf{U}}_{(a,b\rangle}(\phi_1,\phi_2) \;\equiv\; \Diamond_{(a,b\rangle}(\phi_2) \wedge \Box_{(0,a]}\Big(\mathsf{U}_{(0,+\infty)}(\phi_1,\phi_2)\Big) \tag{1}$$

$$\widetilde{\mathsf{U}}_{[a,b\rangle}(\phi_1,\phi_2) \;\equiv\; \widetilde{\mathsf{U}}_{(a,b\rangle}(\phi_1,\phi_2) \vee \Big(\Box_{(0,a)}(\phi_1) \wedge \Diamond_{=a}(\phi_2)\Big) \tag{2}$$

$$\widetilde{\mathsf{U}}_{(0,b\rangle}(\phi_1,\phi_2) \;\equiv\; \Diamond_{(0,b\rangle}(\phi_2) \wedge \widetilde{\bigcirc}\Big(\mathsf{U}_{(0,+\infty)}(\phi_1,\phi_2)\Big) \tag{3}$$

$$\widetilde{\mathsf{U}}_{[0,b\rangle}(\phi_1,\phi_2) \;\equiv\; \widetilde{\mathsf{U}}_{(0,b\rangle}(\phi_1,\phi_2) \vee \phi_2 \tag{4}$$

$$\widetilde{\mathsf{U}}_{[0,0]}(\phi_1,\phi_2) \;\equiv\; \phi_2 \tag{5}$$

(1–5) provide a means to replace each occurrence of strict *until* with non-strict *until*s only. Also, if we replace each occurrence of formula ϕ_2 in (1–5) with $\phi_2 \wedge \phi_1$ — except for (4) which requires a slightly different treatment, which is however routine — we also have a proof that $\widetilde{\text{MTL}}^{\downarrow} = \text{MTL}^{\downarrow}$, according to the definition of the matching variants of the *until* operators.

3.2 Matching as Expressive as Non-matching

This section provides a set of equivalences to replace each occurrence of a strict non-matching operator with a formula that contains only strict matching operators; this shows that $\widetilde{\text{MTL}} = \widetilde{\text{MTL}}^{\downarrow}$. To this end, let us first prove the following equivalence.

$$\begin{aligned}\widetilde{\mathsf{U}}_{(0,b\rangle}(\phi_1,\phi_2) \quad\equiv\quad & \widetilde{\mathsf{U}}^{\downarrow}_{(0,b\rangle}(\phi_1,\phi_2) \\ \vee\; & \big(\Diamond_{(0,b\rangle}(\phi_2) \wedge \widetilde{\bigcirc}(\phi_1) \wedge \widetilde{\mathsf{R}}^{\downarrow}_{(0,b\rangle}(\phi_2, \bigcirc(\phi_1))\end{aligned} \tag{6}$$

Proof (of Formula 6). Let us start with the $\Leftarrow$ direction, and let t be the current instant. If $b(t) \models_{\mathbb{R}} \widetilde{\mathsf{U}}^{\downarrow}_{(0,b\rangle}(\phi_1,\phi_2)$ clearly also $b(t) \models_{\mathbb{R}} \widetilde{\mathsf{U}}_{(0,b\rangle}(\phi_1,\phi_2)$ *a fortiori*. So let us assume that $\widetilde{\mathsf{U}}^{\downarrow}_{(0,b\rangle}(\phi_1,\phi_2)$ is false at t; note that this subsumes that $b(t) \models_{\mathbb{R}} \neg\widetilde{\bigcirc}(\phi_2 \wedge \phi_1)$.

Let us remark that we can assume that $\widetilde{\bigcirc}(\neg\phi_2)$ holds at t, because $\widetilde{\bigcirc}(\phi_1)$ and $\neg\widetilde{\bigcirc}(\phi_2 \wedge \phi_1)$ both hold. Therefore, it is well-defined u, the smallest instant in $(t, t+b\rangle$ such that $b(u) \models_{\mathbb{R}} \phi_2 \vee \widetilde{\bigcirc}(\phi_2)$. Note that this implies that ϕ_2 is false throughout $(t, u\rangle$, with the interval right-open iff ϕ_2 holds at u.

Let us first consider the case $b(u) \models_{\mathbb{R}} \phi_2$. Let v be a generic instant in (t, u); recall that ϕ_2 is false throughout $(t, u) \supset (t, v]$. Therefore it must be $b(v) \models_{\mathbb{R}}$

$\bigcirc(\phi_1)$ for $b(t) \models_{\mathbb{R}} \widetilde{\mathsf{R}}^{\downarrow}_{(0,b\rangle}(\phi_2, \bigcirc(\phi_1))$ to be true. So, ϕ_1 holds throughout (t, u) and ϕ_2 holds at u, which means that $b(t) \models_{\mathbb{R}} \widetilde{\mathsf{U}}_{(0,b\rangle}(\phi_1, \phi_2)$.

Let us now consider the other case $b(u) \models_{\mathbb{R}} \neg\phi_2 \wedge \widetilde{\bigcirc}(\phi_2)$. Let v be a generic instant in $(t, u]$; recall that ϕ_2 is false throughout $(t, u] \supseteq (t, v]$. From $b(t) \models_{\mathbb{R}} \widetilde{\mathsf{R}}^{\downarrow}_{(0,b\rangle}(\phi_2, \bigcirc(\phi_1))$ it must be $b(v) \models_{\mathbb{R}} \bigcirc(\phi_1)$. Overall, ϕ_1 holds throughout $(t, u+\epsilon]$ for some $\epsilon > 0$, as in particular $b(t+u) \models_{\mathbb{R}} \bigcirc(\phi_1)$. Clearly, this subsumes $b(t) \models_{\mathbb{R}} \widetilde{\mathsf{U}}_{(0,b\rangle}(\phi_1, \phi_2)$.

For brevity, we omit the simpler $\Rightarrow$ direction (see [17] for details). □

The case for $a > 0$ can be handled simply by relying on the previous equivalence. In fact, the following equivalence is easily seen to hold.

$$\begin{aligned} \widetilde{\mathsf{U}}_{(a,b\rangle}(\phi_1, \phi_2) \quad &\equiv \quad \widetilde{\mathsf{U}}^{\downarrow}_{(a,b\rangle}(\phi_1, \phi_2) \\ &\quad \vee \left(\Box_{(0,a]}(\phi_1) \wedge \Diamond_{=a}\left(\widetilde{\mathsf{U}}_{(0,b-a\rangle}(\phi_1, \phi_2)\right)\right) \end{aligned} \tag{7}$$

The cases for left-closed intervals are also derivable straightforwardly as:

$$\widetilde{\mathsf{U}}_{[0,b\rangle}(\phi_1, \phi_2) \quad \equiv \quad \phi_2 \vee \widetilde{\mathsf{U}}_{(0,b\rangle}(\phi_1, \phi_2) \tag{8}$$

and

$$\widetilde{\mathsf{U}}_{[a,b\rangle}(\phi_1, \phi_2) \quad \equiv \quad \left(\Diamond_{=a}(\phi_2) \wedge \Box_{(0,a)}(\phi_1)\right) \vee \widetilde{\mathsf{U}}_{(a,b\rangle}(\phi_1, \phi_2) \tag{9}$$

Finally, let us note that the $\bigcirc(\phi)$ operator can be expressed equivalently with strict matching operators as $\phi \wedge \widetilde{\mathsf{U}}^{\downarrow}_{(0,+\infty)}(\phi, \top)$. In fact, $\bigcirc(\phi)$ at x means that ϕ holds over an interval $[x, x+\epsilon)$ for some $\epsilon > 0$; therefore, ϕ also holds over a closed interval such as $[x, x+\epsilon/2]$, as required by $\phi \wedge \widetilde{\mathsf{U}}^{\downarrow}_{(0,+\infty)}(\phi, \top)$, and *vice versa*.

All in all (6–9) provide a means to replace every occurrence of strict non-matching *until* with a formula that contains only strict matching *until*s. This shows that $\widetilde{\mathrm{MTL}} = \widetilde{\mathrm{MTL}}^{\downarrow}$, completing our set of equivalences for non-Zeno behaviors.

4 Flat MTL

Section 3 has shown the equivalence of all (non-)strict and (non-)matching MTL variants for non-Zeno behaviors. It is apparent, however, that the equivalences between the various *until* variants introduce nesting of temporal operators, that is they change flat formulas into nesting ones. This section shows that this is inevitable, as the relative expressiveness relations change if we consider flat formulas only. More precisely, we prove that both non-strictness and matchingness lessen the expressive power of MTL flat formulas, so that the strict non-matching variant is shown to be the most expressive. We also show that, as one would expect, even this most expressive flat variant is less expressive than any nesting variant. All separation results are proved under both the initial satisfiability and the global satisfiability semantics.

4.1 Non-strict Less Expressive Than Strict

This section shows that $\flat\text{MTL} \subset \flat\widetilde{\text{MTL}}$; let us outline the technique used to prove this fact. We provide a strict flat formula $\alpha \in \flat\widetilde{\text{MTL}}$ and we prove that it has no equivalent non-strict flat formula. The proof goes adversarially: assume $\beta \in \flat\text{MTL}$ is a non-strict flat formula equivalent to α, and let ρ be the granularity of β. From ρ we build two behaviors $b^\rho_\top$ and $b^\rho_\bot$ such that any $\flat$MTL formula of granularity ρ (and β in particular) cannot distinguish between them, i.e., it is either satisfied by both or by none. On the contrary, α is satisfied by $b^\rho_\top$ but not by $b^\rho_\bot$, for all ρ. This shows that no equivalent non-strict flat formula can exist, and thus $\flat\widetilde{\text{MTL}} \not\subseteq \flat\text{MTL}$. From $\flat\text{MTL} \subseteq \flat\widetilde{\text{MTL}}$ we conclude that $\flat\text{MTL} \subset \flat\widetilde{\text{MTL}}$.

As in all separation results, the details of the proofs are rather involved; this is even more the case when considering the global satisfiability semantics; throughout we will try to provide some intuition referring to [17] for all the lower-level details.

Let us define the following families of behaviors over $\{\mathsf{p}\}$. For any given $\rho > 0$, let $b^\rho_\top$ and $b^\rho_\bot$ be defined as follows: $\mathsf{p} \in b^\rho_\top(t)$ iff $t \leq 0$ or $t \geq \rho/4$; and $\mathsf{p} \in b^\rho_\bot(t)$ iff $t \leq 0$ or $t > \rho/4$. Similarly, for any given $\rho > 0$, $c^\rho_\bot$ is defined as follows: $\mathsf{p} \in c^\rho_\bot(t)$ iff $\mathsf{p} \in b^\rho_\bot(t)$ and $t \neq 0$. Note that $b^\rho_\top(t) = b^\rho_\bot(t)$ for all $t \neq \rho/4$, and that $b^\rho_\bot(t) = c^\rho_\bot(t)$ for all $t \neq 0$. Let us also define the sets of behaviors $\mathcal{B}_\top = \bigcup_{\rho \in \mathbb{Q}_{>0}} b^\rho_\top$ and $\mathcal{B}_\bot = \bigcup_{\rho \in \mathbb{R}_{>0}} b^\rho_\bot$. The behaviors $b^\rho_\top, b^\rho_\bot, c^\rho_\bot$ are pictured in Figure 1.

Initial satisfiability. Let us first assume the initial satisfiability semantics; we show that no $\flat$MTL formula ϕ with granularity ρ distinguishes initially between $b^\rho_\top$ and $b^\rho_\bot$. To this end, we prove the following.

Lemma 1. *For any* $\flat$MTL *formula* ϕ *of granularity* ρ*, it is* $b^\rho_\top(0) \models_{\mathbb{R}} \phi$ *iff* $b^\rho_\bot(0) \models_{\mathbb{R}} \phi$.

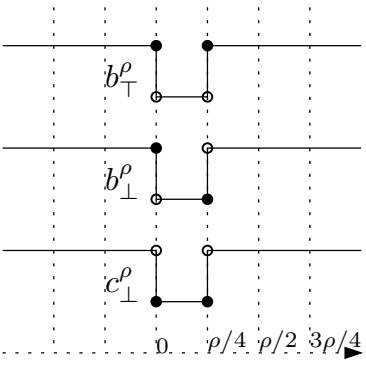


Fig. 1. The behaviors $b^\rho_\top, b^\rho_\bot, c^\rho_\bot$

Proof. The proof is by induction on the structure of ϕ. Let us consider just a few most relevant cases (the others are in [17]): assume $\phi = \mathsf{U}_I(\beta_1, \beta_2)$, with $I = \langle l, u\rangle$ and: (1) $l = k_1\rho$ for some $k_1 \in \mathbb{N}$; and (2) $u = k_2\rho$ or $u = +\infty$, for some $k_1 \leq k_2 \in \mathbb{N}$. Note that we can assume $0 \notin I$ without loss of generality, as $\mathsf{U}_{[0,u\rangle}(\beta_1, \beta_2) \equiv \beta_2 \vee \mathsf{U}_{(0,u\rangle}(\beta_1, \beta_2)$. We also assume that I is non-empty; this is also without loss of generality. We then consider all cases for β_1, β_2; in particular:

- If $\beta_1 \equiv \top$ and β_2 is one of p or $\neg\mathsf{p}$, we have $\phi \equiv \Diamond_I(\beta_2)$.
  If $l = 0$, then $u > l$, which entails $u \geq \rho$. Any interval of the form $\langle 0, \rho\rangle$ encompasses both instants where p holds and instants where $\neg\mathsf{p}$ holds. Thus, $b^\rho_\top(0) \models_{\mathbb{R}} \phi$ and $b^\rho_\bot(0) \models_{\mathbb{R}} \phi$ in this case.
  If $l > 0$ then $l \geq \rho$. Then, whatever $u \geq l$ is, it is clear that p holds throughout the non-empty interval $\langle l, u\rangle$. Therefore, if $\beta_2 \equiv \mathsf{p}$ we have $b^\rho_\top(0) \models_{\mathbb{R}} \phi$ and $b^\rho_\bot(0) \models_{\mathbb{R}} \phi$; otherwise $\beta_2 \equiv \neg\mathsf{p}$, and $b^\rho_\top(0) \not\models_{\mathbb{R}} \phi$ and $b^\rho_\bot(0) \not\models_{\mathbb{R}} \phi$.
- If $\beta_1 \equiv \mathsf{p}$ or $\beta_1 \equiv \neg\mathsf{p}$, then ϕ does not hold unless $\bigcirc(\beta_1)$ holds (still because $0 \notin I$). Since the value of p changes from 0 to its immediate future, it is $b^\rho_\top(0) \not\models_{\mathbb{R}} \phi$ and $b^\rho_\bot(0) \not\models_{\mathbb{R}} \phi$. □

Lemma 1 leads straightforwardly to the desired separation result.

Theorem 1. *Under the initial satisfiability semantics,* $\flat\mathrm{MTL} \subset \flat\widetilde{\mathrm{MTL}}$.

Proof. Let us show that the $\flat\widetilde{\mathrm{MTL}}$ formula $\Sigma = \widetilde{\mathsf{U}}_{(0,+\infty)}(\neg\mathsf{p}, \mathsf{p})$ has no equivalent $\flat$MTL formula. Σ can distinguish initially between the families of behaviors $\mathcal{B}_\top, \mathcal{B}_\bot$, as for any $b \in \mathcal{B}_\top, b' \in \mathcal{B}_\bot$, it is $b(0) \models_{\mathbb{R}} \Sigma$ and $b'(0) \not\models_{\mathbb{R}} \Sigma$. Let us assume that σ is an $\flat$MTL formula of granularity ρ equivalent to Σ. However, from Lemma 1 it follows that $b^\rho_\top(0) \models_{\mathbb{R}} \sigma$ iff $b^\rho_\bot(0) \models_{\mathbb{R}} \sigma$. Therefore, σ is not equivalent to Σ. □

Global satisfiability. Let us now prove an analogous of Theorem 1 for the global satisfiability semantics.

Lemma 2. *For any* $\flat$MTL *formula* ϕ *of granularity* ρ *and any instant* $t < 0$*, it is:* $b^\rho_\top(t) \models_{\mathbb{R}} \phi$ *iff* $b^\rho_\bot(t) \models_{\mathbb{R}} \phi$*, or* $b^\rho_\bot(t) \models_{\mathbb{R}} \phi$ *iff* $c^\rho_\bot(t) \models_{\mathbb{R}} \phi$*.*

Proof. The proof is by induction on the structure of ϕ; throughout, t is any fixed instant less than 0. Let us just outline a few significant cases; all details are in [17].

For the base case, if $\phi = \mathsf{U}_I(\beta_1, \beta_2)$ with $\beta_1, \beta_2 \in \{\mathsf{p}, \neg\mathsf{p}, \top, \bot\}$ one can verify that it is $b^\rho_\top \models_{\mathbb{R}} \phi$ iff $b^\rho_\bot \models_{\mathbb{R}} \phi$ iff $c^\rho_\bot \models_{\mathbb{R}} \phi$, unless: (a) $b^\rho_\top \not\models_{\mathbb{R}} \phi$ iff $b^\rho_\bot \models_{\mathbb{R}} \phi$ iff $c^\rho_\bot \models_{\mathbb{R}} \phi$ and $t + k\rho = \rho/4$ or $t + k\rho = 0$ for some positive integer k; or (b) $b^\rho_\top \models_{\mathbb{R}} \phi$ iff $b^\rho_\bot \models_{\mathbb{R}} \phi$ iff $c^\rho_\bot \not\models_{\mathbb{R}} \phi$ and $t + h\rho = 0$ for some positive integer h.

Therefore, consider the inductive case $\phi = \phi_1 \wedge \phi_2$; in particular let us focus on the "crucial" case $b^\rho_\top(t) \not\models_{\mathbb{R}} \phi_i$ iff $b^\rho_\bot(t) \models_{\mathbb{R}} \phi_i$ iff $c^\rho_\bot(t) \models_{\mathbb{R}} \phi_i$ and $t + k\rho = \rho/4$ for some i, and $b^\rho_\top(t) \models_{\mathbb{R}} \phi_j$ iff $b^\rho_\bot(t) \models_{\mathbb{R}} \phi_j$ iff $c^\rho_\bot(t) \not\models_{\mathbb{R}} \phi_j$ and $t + h\rho = 0$ for $j \neq i$. This case, however, is not possible as $t + k\rho = \rho/4 = \rho/4 + t + h\rho$ implies $(k - h)\rho = \rho/4$ which is impossible as k and h are integers. To give

some intuition, this is due to the granularity: in other words, from the same t we cannot reference both 0 and $\rho/4$, since they are less than ρ time instants apart. Finally also note that this restriction can be "lifted" to the conjunction itself, to go with the inductive hypothesis. □

Through Lemma 2 we can extend Theorem 1 to the global satisfiability semantics.

Theorem 2. *Under the global satisfiability semantics,* $\flat\mathrm{MTL} \subset \flat\widetilde{\mathrm{MTL}}$.

Proof. Let us show that the $\flat\widetilde{\mathrm{MTL}}$ formula $\Omega = \widetilde{\mathsf{U}}_{(0,+\infty)}(\neg\mathsf{p}, \mathsf{p}) \vee \bigcirc(\neg\mathsf{p}) \vee \widetilde{\bigcirc}(\mathsf{p})$ has no equivalent $\flat\mathrm{MTL}$ formula. It is simple to check that, for all $b \in \mathcal{B}_\top, b' \in \mathcal{B}_\bot$ it is $b \models_{\mathbb{R}} \Omega$ and $b' \not\models_{\mathbb{R}} \Omega$; more precisely, it is $b'(0) \not\models_{\mathbb{R}} \Omega$ and, for all $t > 0$, $b'(t) \models_{\mathbb{R}} \Omega$. Also, for all $b'' \in \bigcup_\rho c^\rho_\bot$, it is $b'' \models_{\mathbb{R}} \Omega$.

Now the proof goes by *reductio ad absurdum*. Let ω be an $\flat\mathrm{MTL}$ formula of granularity ρ equivalent to Ω. Thus it must be $b^\rho_\top \models_{\mathbb{R}} \omega$, $b^\rho_\bot \not\models_{\mathbb{R}} \omega$, and $c^\rho_\bot \models_{\mathbb{R}} \omega$. So, there exists a t such that $b^\rho_\bot(t) \not\models_{\mathbb{R}} \omega$. Let us show that no such t can exist. Lemma 1 mandates that $b^\rho_\bot(0) \models_{\mathbb{R}} \omega$, so it must be $t \neq 0$.

If $t > 0$, recall that $c^\rho_\bot \models_{\mathbb{R}} \omega$. This subsumes that $c^\rho_\bot(u) \models_{\mathbb{R}} \omega$ for all $u > 0$, and thus in particular at t. However, note that $c^\rho_\bot(x) = b^\rho_\bot(x)$ for all $x > 0$; since ω is a future formula, its truth value to the future of 0 cannot change when just one *past* instant has changed and the future has not changed. So it must be $b^\rho_\bot(t) \models_{\mathbb{R}} \omega$: a contradiction.

Let us now assume $t < 0$. From Lemma 2 for formula ω, it is either (1) $b^\rho_\top(t) \models_{\mathbb{R}} \omega$ iff $b^\rho_\bot(t) \models_{\mathbb{R}} \omega$; or (2) $b^\rho_\bot(t) \models_{\mathbb{R}} \omega$ iff $c^\rho_\bot(t) \models_{\mathbb{R}} \omega$. However, $b^\rho_\top(t) \models_{\mathbb{R}} \omega$ and $b^\rho_\bot(t) \not\models_{\mathbb{R}} \omega$, so (1) is false and (2) must be true. Hence, it must be $c^\rho_\bot(t) \not\models_{\mathbb{R}} \omega$. But this implies $c^\rho_\bot \not\models_{\mathbb{R}} \omega$, whereas it should be $c^\rho_\bot \models_{\mathbb{R}} \omega$ since ω is supposed equivalent to Ω: a contradiction again. □

4.2 Matching Less Expressive Than Non-matching

This section provides an indirect simple proof that $\flat\widetilde{\mathrm{MTL}}^\downarrow \subset \flat\widetilde{\mathrm{MTL}}$. To this end we first show the equivalence of non-strict and strict flat matching MTL when restricted to a unary set of propositions.

Lemma 3. *Over a unary set of propositions* $\mathcal{P} : |\mathcal{P}| = 1$, $\flat\mathrm{MTL}^\downarrow = \flat\widetilde{\mathrm{MTL}}^\downarrow$.

Proof (sketch). We can show that any $\flat\widetilde{\mathrm{MTL}}^\downarrow$ formula $\phi = \widetilde{\mathsf{U}}^\downarrow_I(\beta_1, \beta_2)$ has an equivalent $\flat\mathrm{MTL}^\downarrow$ formula for unary alphabet, as when $\beta_1 = \neg\beta_2$ ϕ is trivially false, according to the semantics of Section 2. □

As a corollary of Lemma 3 we can separate $\flat\widetilde{\mathrm{MTL}}^\downarrow$ and $\flat\widetilde{\mathrm{MTL}}$ (over general set of propositions).

Theorem 3. $\flat\widetilde{\mathrm{MTL}}^\downarrow \subset \flat\widetilde{\mathrm{MTL}}$.

Proof. Recall that $\flat\text{MTL}^{\downarrow} \subseteq \flat\text{MTL}$. Let us first assume a unary alphabet; from Lemma 3 it is $\flat\widetilde{\text{MTL}}^{\downarrow} = \flat\text{MTL}^{\downarrow}$. Since Theorems 1 and 2 are based on counterexamples over unary alphabet, it is also $\flat\text{MTL} \subset \flat\widetilde{\text{MTL}}$. All in all: $\flat\widetilde{\text{MTL}}^{\downarrow} = \flat\text{MTL}^{\downarrow} \subseteq \flat\text{MTL} \subset \flat\widetilde{\text{MTL}}$, hence $\flat\widetilde{\text{MTL}}^{\downarrow} \subset \flat\widetilde{\text{MTL}}$ over unary alphabet, which implies the same holds over generic alphabet. □

4.3 Non-strict Matching Less Expressive Than Matching

Section 4.1 shows that strict flat MTL is strictly more expressive than its non-strict flat counterpart, when both of them are in their non-matching version. If we consider the matching versions of strict and non-strict operators, one can prove that the same holds, that is $\flat\text{MTL}^{\downarrow} \subset \flat\widetilde{\text{MTL}}^{\downarrow}$.

Lemma 3 entails that any separation proofs for $\flat\text{MTL}^{\downarrow}$ and $\flat\widetilde{\text{MTL}}^{\downarrow}$ must consider behaviors over alphabets of size at least two. In fact, it is possible to use a technique similar to that of Section 4.1, but with behaviors over alphabet of size two. For details we refer to [17].

Theorem 4. *Under the initial and global satisfiability semantics,* $\flat\text{MTL}^{\downarrow} \subset \flat\widetilde{\text{MTL}}^{\downarrow}$.

4.4 Non-strict Matching Less Expressive Than Non-strict

Section 4.2 shows that flat non-matching MTL is strictly more expressive than its matching flat counterpart, when both of them are in their strict version. The same relation holds if we consider the non-strict versions of matching and non-matching operators, that is we can prove that $\flat\text{MTL}^{\downarrow} \subset \flat\text{MTL}$. The proof technique is again is similar to the one in the previous Section 4.3, and it is based on behaviors over a binary set of propositions; see [17] for details.

Theorem 5. *Under the initial and global satisfiability semantics,* $\flat\text{MTL}^{\downarrow} \subset \flat\text{MTL}$.

4.5 Flat Less Expressive Than Nesting

Through a technique similar to that used in Section 4.1 it is also possible to show that $\flat\widetilde{\text{MTL}} \subset \text{MTL}$. For the lack of space we refer to [17] for all details.

Theorem 6. *Under the initial and global satisfiability semantics,* $\flat\widetilde{\text{MTL}} \subset \text{MTL}$.

5 Nesting MTL over Zeno Behaviors

This section re-considers some of the expressiveness results for nesting formulas of Section 3 when Zeno behaviors are allowed as interpretation structures, and in particular it shows that the equivalence between MTL and $\widetilde{\text{MTL}}$ does not hold if we allow Zeno behaviors.

5.1 Non-strict Less Expressive Than Strict

Let us first show that MTL $\subset \widetilde{\text{MTL}}$ over generic behaviors. To this end, we define behaviors $b_\delta, b_\delta^{\mathcal{Z}}$ over $\mathcal{P} = \{\mathsf{p}\}$, for all $\delta > 0$. b_δ is defined as: $\mathsf{p} \in b_\delta(t)$ iff $t = k\delta/2$ for some $k \in \mathbb{Z}$. $b_\delta^{\mathcal{Z}}$ is defined as: $\mathsf{p} \in b_\delta^{\mathcal{Z}}(t)$ iff $t = (k + 2^{-n})\delta$, for some $k \in \mathbb{Z}$, $n \in \mathbb{N}$. Clearly, for all $t \in \mathbb{T}$, $\mathsf{p} \in b_\delta(t)$ implies $\mathsf{p} \in b_\delta^{\mathcal{Z}}(t)$; moreover, notice that $b_\delta^{\mathcal{Z}}$ has Zeno behavior to the right of any instant $k\delta$.

Through the usual case analysis on the structure of formulas, we can prove that the behavior of any MTL formula over b_δ and $b_\delta^{\mathcal{Z}}$ is very simple, as it coincides with one of $\mathsf{p}, \neg\mathsf{p}, \top, \bot$ (see [17] for all details).

Lemma 4. *The truth value of any* MTL *formulas ϕ of granularity δ coincides with one of $\mathsf{p}, \neg\mathsf{p}, \top, \bot$ over both b_δ and $b_\delta^{\mathcal{Z}}$.*

An immediate consequence of the previous lemma is that, at any instant where the values $b_\delta(t)$ and $b_\delta^{\mathcal{Z}}(t)$ coincide, the truth values of any formula ϕ also coincide.

Corollary 1. *For any* MTL *formula ϕ of granularity δ, and all $k \in \mathbb{Z}$: $b_\delta(k\delta) \models_{\mathbb{R}} \phi$ iff $b_\delta^{\mathcal{Z}}(k\delta) \models_{\mathbb{R}} \phi$.*

Finally, we prove the desired separation result as follows.

Theorem 7. *If Zeno behaviors are allowed,* MTL $\subset \widetilde{\text{MTL}}$.

Proof. Let us consider the two families of behaviors: $\mathcal{N} = \{b_\delta \mid \delta \in \mathbb{Q}_{>0}\}$ and $\mathcal{Z} = \{b_\delta^{\mathcal{Z}} \mid \delta \in \mathbb{Q}_{>0}\}$.

First, let us consider initial satisfiability. The $\widetilde{\text{MTL}}$ formula $\Sigma = \widetilde{\bigcirc}(\neg\mathsf{p})$ separates initially the two families $\mathcal{N}$ and $\mathcal{Z}$, as $b(0) \models_{\mathbb{R}} \Sigma$ for all $b \in \mathcal{N}$ and $b'(0) \not\models_{\mathbb{R}} \Sigma$ for all $b' \in \mathcal{Z}$.

On the contrary, let ϕ be any MTL formula, and let δ be its granularity. Then, $\mathcal{N} \ni b_\delta(0) \models_{\mathbb{R}} \phi$ iff $\mathcal{Z} \ni b_\delta^{\mathcal{Z}}(0) \models_{\mathbb{R}} \phi$ by Corollary 1, so no MTL formula separates initially the two families. This implies that the $\widetilde{\text{MTL}}$ formula Σ has no initially equivalent formula in MTL.

Now, let us consider global satisfiability. The $\widetilde{\text{MTL}}$ formula $\Sigma' = \mathsf{p} \Rightarrow \widetilde{\bigcirc}(\neg\mathsf{p})$ separates globally the two families $\mathcal{N}$ and $\mathcal{Z}$, as $b(t) \models_{\mathbb{R}} \Sigma'$ for all $t \in \mathbb{T}$ and for all $b \in \mathcal{N}$, and $b'(t) \not\models_{\mathbb{R}} \Sigma'$ for some $t = k\delta$, and for all $b' \in \mathcal{Z}$.

On the contrary, let ϕ be any MTL formula, and let δ be its granularity. For the sake of contradiction, assume that $b(t) \models_{\mathbb{R}} \phi$ for all $t \in \mathbb{T}$ and for all $b \in \mathcal{N}$, and that $b'(t) \not\models_{\mathbb{R}} \phi$ for some t, and for all $b' \in \mathcal{Z}$. Now, in particular, $b_\delta \models_{\mathbb{R}} \phi$; *a fortiori*, $b_\delta(0) \models_{\mathbb{R}} \phi'$ where $\phi' = \Box_{[0,+\infty)}(\phi)$. Similarly, it must be $b_\delta^{\mathcal{Z}} \not\models_{\mathbb{R}} \phi$. A little reasoning should convince us that this implies $b_\delta^{\mathcal{Z}}(0) \not\models_{\mathbb{R}} \phi'$. In fact, $b_\delta^{\mathcal{Z}} \not\models_{\mathbb{R}} \phi$ means that there exists a $t \in \mathbb{T}$ such that $b_\delta^{\mathcal{Z}}(t) \not\models_{\mathbb{R}} \phi$. t may be greater than, equal to, or less than 0. However, $b_\delta^{\mathcal{Z}}$ is *periodic* with period δ; this implies that $b_\delta^{\mathcal{Z}}(t) \models_{\mathbb{R}} \alpha$ iff $b_\delta^{\mathcal{Z}}(t + k\delta) \models_{\mathbb{R}} \alpha$, for all formulas α, $t \in \mathbb{T}$, $k \in \mathbb{Z}$. Therefore, if there exists a $t \in \mathbb{T}$ such that $b_\delta^{\mathcal{Z}}(t) \not\models_{\mathbb{R}} \phi$, then also there exists a $t' \geq 0$ such that $b_\delta^{\mathcal{Z}}(t') \not\models_{\mathbb{R}} \phi$. The last formula implies that $b_\delta^{\mathcal{Z}}(0) \not\models_{\mathbb{R}} \phi'$.

Now, notice that the formula ϕ' is of the same granularity as ϕ, that is δ. Moreover, $b_\delta(0) \models_{\mathbb{R}} \phi'$ and $b_\delta^{\mathbb{Z}}(0) \not\models_{\mathbb{R}} \phi'$. This contradicts Corollary 1; therefore ϕ does not globally separate the two families of behaviors. Since ϕ is generic, the $\widetilde{\text{MTL}}$ formula Σ' has no globally equivalent formula in MTL. □

Finally, if we reconsider all the theorems of the current section, and the corresponding proofs, we notice that they still stand for the matching variants of the non-strict and strict *until*. In other words, the same proofs provide a separation between $\text{MTL}^{\downarrow}$ and $\widetilde{\text{MTL}}^{\downarrow}$.

5.2 Matching as Expressive as Non-matching

A careful reconsideration of the proofs of (6–9) shows that the equivalences hold even when Zeno behaviors are allowed; essentially, Zeno behaviors can be explicitly dealt with in the proof.[4] In summary, we have a proof that $\widetilde{\text{MTL}} = \widetilde{\text{MTL}}^{\downarrow}$ even if Zeno behaviors are allowed. As an example, let us sketch a bit of the proof of (6) for Zeno behaviors.

Proof (of (6) for Zeno behaviors). For the $\Leftarrow$ direction, let us consider the case: $b(t) \models_{\mathbb{R}} \neg\widetilde{\bigcirc}(\phi_2 \wedge \phi_1)$ and $b(t) \models_{\mathbb{R}} \widetilde{\bigcirc}(\phi_1)$, i.e., ϕ_1 holds over an interval $(t, t+\epsilon)$ for some $\epsilon > 0$. If ϕ_2 has Zeno behavior to the right of t, it changes truth value infinitely many times over $\min(\epsilon, b)$. Hence, there exists a $0 < \nu < \min(\epsilon, b)$ such that $b(t+\nu) \models_{\mathbb{R}} \phi_2$; so $b(t) \models_{\mathbb{R}} \widetilde{\mathsf{U}}_{(0,b\rangle}(\phi_1, \phi_2)$ *a fortiori*. The other cases are done similarly (see [17]). □

Furthermore, it is possible to adapt (6–9) to use non-strict operators only. In practice, (7–9) hold if we just replace strict operators with the corresponding non-strict ones; on the other hand, (6) should be modified as:

$$\mathsf{U}_{(0,b\rangle}(\phi_1, \phi_2) \equiv \mathsf{U}^{\downarrow}_{(0,b\rangle}(\phi_1, \phi_2) \vee (\Diamond_{(0,b\rangle}(\phi_2) \wedge \bigcirc(\phi_1) \wedge \mathsf{R}^{\downarrow}_{(0,b\rangle}(\phi_2 \wedge \neg\phi_1, \bigcirc(\phi_1)) \tag{10}$$

All the resulting new equivalences using only non-strict operators can be shown to hold for Zeno, as well as non-Zeno, behaviors (see [17]). Hence, we have a proof that $\text{MTL} = \text{MTL}^{\downarrow}$ over generic behaviors.

6 Summary and Discussion

Figure 2 displays the relative expressiveness relations for non-Zeno behaviors (left) and Zeno behaviors (right). Note that the separation proofs for the flat fragments used only non-Zeno behaviors, therefore they imply the separation of the corresponding classes for generic (i.e., including Zeno) behaviors as well. On the other hand, the problem of the relative expressiveness of $\flat$MTL and $\widetilde{\flat\text{MTL}}^{\downarrow}$ is currently open (over both Zeno and non-Zeno behaviors).

[4] We are grateful to anonymous referee #4, whose detailed comments prompted us to realize this fact.

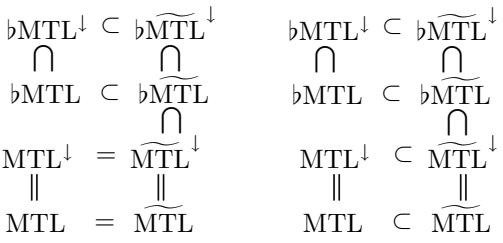


Fig. 2. Expressiveness over non-Zeno (left) and Zeno (right) behaviors

Tackling this open question about MTL relative expressiveness, and considering other variations such as the use of past operators, belongs to future work.

Acknowledgements. We thank the anonymous referees, especially #4, for their scrupulous and relevant observations that contributed to improve our work.

References

1. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM 43(1), 116–146 (1996)
2. Alur, R., Henzinger, T.A.: Back to the future. In: Proceedings of FOCS'92, pp. 177–186 (1992)
3. Alur, R., Henzinger, T.A.: Logics and models of real time. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) Real-Time: Theory in Practice. LNCS, vol. 600, pp. 74–106. Springer, Heidelberg (1992)
4. Alur, R., Henzinger, T.A.: Real-time logics: Complexity and expressiveness. Information and Computation 104(1), 35–77 (1993)
5. Bouyer, P., Chevalier, F., Markey, N.: On the expressiveness of TPTL and MTL. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 432–443. Springer, Heidelberg (2005)
6. Bouyer, P., Markey, N., Ouaknine, J., Worrell, J.: The cost of punctuality. In: Proceedings of LICS'07 (2007)
7. Comon, H., Cortier, V.: Flatness is not a weakness. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 262–276. Springer, Heidelberg (2000)
8. Dams, D.: Flat fragments of CTL and CTL*: Separating the expressive and distinguishing powers. Logic Journal of the IGPL 7(1), 55–78 (1999)
9. Demri, S., Schnoebelen, P.: The complexity of propositional linear temporal logics in simple cases. Information and Computation 174(1), 84–103 (2002)
10. D'Souza, D., Mohan, R.M., Prabhakar, P.: Flattening metric temporal logic. Manuscript (2007)
11. D'Souza, D., Prabhakar, P.: On the expressiveness of MTL in the pointwise and continuous semantics. Formal Methods Letters 9(1), 1–4 (2007)
12. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 996–1072. Elsevier, Amsterdam (1990)
13. Etessami, K., Wilke, T.: An until hierarchy for temporal logic. In: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS'96), pp. 108–117. IEEE Computer Society Press, Los Alamitos (1996)

14. Furia, C.A.: Scaling Up the Formal Analysis of Real-Time Systems. PhD thesis, Dipartimento di Elettonica e Informazione, Politecnico di Milano (May 2007)
15. Furia, C.A., Rossi, M.: Integrating discrete- and continuous-time metric temporal logics through sampling. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 215–229. Springer, Heidelberg (2006)
16. Furia, C.A., Rossi, M.: No need to be strict. Bulletin of the EATCS (to appear, 2007)
17. Furia, C.A., Rossi, M.: On the expressiveness of MTL variants over dense time. Technical Report 2007.41, DEI, Politecnico di Milano (May 2007)
18. Gabbay, D.M., Hodkinson, I., Reynolds, M.: Temporal Logic, vol. 1. Oxford University Press, Oxford (1994)
19. Gargantini, A., Morzenti, A.: Automated deductive requirement analysis of critical systems. ACM TOSEM 10(3), 255–307 (2001)
20. Hirshfeld, Y., Rabinovich, A.: Logics for real time. Fundamenta Informaticae 62(1), 1–28 (2004)
21. Hirshfeld, Y., Rabinovich, A.: Expressiveness of metric modalities for continuous time. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 211–220. Springer, Heidelberg (2006)
22. Hirshfeld, Y., Rabinovich, A.M.: Future temporal logic needs infinitely many modalities. Information and Computation 187(2), 196–208 (2003)
23. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems 2(4), 255–299 (1990)
24. Kučera, A., Strejček, J.: The stuttering principle revisited. Acta Informatica 41(7/8), 415–434 (2005)
25. Laroussinie, F., Markey, N., Schnoebelen, P.: Temporal logic with forgettable past. In: Proceedings of LICS'02, pp. 383–392. IEEE Computer Society Press, Los Alamitos (2002)
26. 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)
27. Maler, O., Nickovic, D., Pnueli, A.: From MITL to timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 274–289. Springer, Heidelberg (2006)
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. 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)
30. Rabinovich, A.M.: Automata over continuous time. Theoretical Computer Science 300(1–3), 331–363 (2003)
31. Reynolds, M.: The complexity of the temporal logic with until over general linear time. Journal of Computer and System Sciences 66(2), 393–426 (2003)
32. Thérien, D., Wilke, T.: Nesting until and since in linear temporal logic. Theory of Computing Systems 37(1), 111–131 (2004)

Quantitative Model Checking Revisited: Neither Decidable Nor Approximable*

Sergio Giro and Pedro R. D'Argenio

FaMAF, Universidad Nacional de Córdoba - CONICET
Ciudad Universitaria - 5000 Córdoba - Argentina
{sgiro,dargenio}@famaf.unc.edu.ar

Abstract. Quantitative model checking computes the probability values of a given property quantifying over all possible schedulers. It turns out that maximum and minimum probabilities calculated in such a way are overestimations on models of distributed systems in which components are loosely coupled and share little information with each other (and hence arbitrary schedulers may result too powerful). Therefore, we focus on the quantitative model checking problem restricted to *distributed* schedulers that are obtained only as a combination of *local* schedulers (i.e. the schedulers of each component) and show that this problem is undecidable. In fact, we show that there is no algorithm that can compute an approximation to the maximum probability of reaching a state within a given bound when restricted to distributed schedulers.

1 Introduction

The model of Markov decision processes (MDP) [14] is a well-known formalism to study systems in which both probabilistic and nondeterministic choices interact. They are used in such diverse fields as operation research, ecology, economics, and computer science. In particular, MDPs (specially composition oriented versions like probabilistic automata [15] or probabilistic modules [7]) are useful to model and analyze concurrent systems such as distributed systems, and serve as the input model to succesful quantitative model checkers such as PRISM [9].

Analysis techniques for MDPs require to consider the resolution of all nondeterministic choices in order to obtain the desired result. For instance, one may like to use PRISM to find out which is the best probability value of reaching a goal under any possible resolution of the nondeterminism (a concrete instance being "the probability of reaching an error state is below the bound 0.01"). The resolution of such nondeterminism is given by the so called *schedulers* (called also adversaries or policies, see [14,1,15,5,17]). A scheduler is a function mapping traces to transitions or *moves* (or, in the more general case, traces to distributions on moves). Given the nondeterministic moves available at some state, the

* Supported by the CONICET/CNRS Cooperation project "Métodos para la Verificación de Programas Concurrentes con aspectos Aleatorios y Temporizados", ANPCyT project PICT 26135 and CONICET project PIP 6391. The first author was partially supported by the LSIS, UMR CNRS 6168, France.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 179–194, 2007.

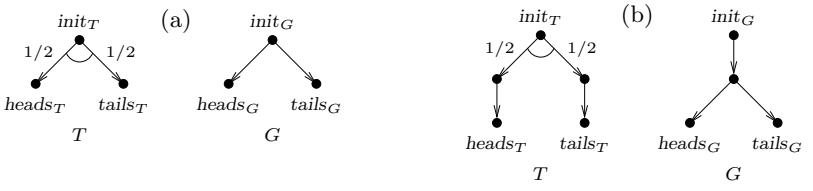


Fig. 1. T tosses a coin and G has to guess

scheduler chooses the move to perform based on the history until the actual state. Then, by quantifying over all the possible schedulers, we obtain maximal and minimal probabilities for (sets of) traces. Quantitative model checkers such as [9,10,4] are based on the technique introduced in [1], in which calculations are performed considering the set of *all* possible schedulers.

While this approach arises naturally, and it captures the intuitive notion of considering all the possible combinations of choices, this arbitrary type of schedulers may yield unexpected results. Consider the following example: a man tosses a coin and another one has to guess heads or tails. Fig. 1(a) depicts the models of these men in terms of MDPs. Man T, who tosses the coin, has only one move which represents the toss of the coin: with probability $1/2$ moves to state $heads_T$ and with probability $1/2$ moves to state $tails_T$. Instead, man G has two nondeterministic moves each one representing his choice: $heads_G$ or $tails_G$. An *almighty* scheduler for this system may let G guess the correct answer with probability 1 according to the following sequence: first, it lets T toss the coin, and then it chooses for G the left move if T tossed a head or the right move if T tossed a tail. Therefore, the maximum probability of guessing obtained by quantifying over these almighty schedulers is 1, even if T is a smart player that always hides the outcome until G reveals his choice. In this example, in which T and G do not share all information, we would like that the maximum probability of guessing (i.e., of reaching any of the states $(heads_T, heads_G)$ or $(tails_T, tails_G)$) is $1/2$. This observation is fundamental in distributed systems in which components are loosely coupled and share little information with each other. (We remark that a similar example is obtained under a synchronous point of view where G performs an idling move when T tosses the coin, and T idles when G performs his choice. See Fig. 1(b).)

This phenomenon has been first observed in [15] from the point of view of compositionality and studied in [6,7,3,2,19] in many different settings, but none of them aims for automatic analysis or verification with the exception of [6], in which temporal properties are quantified over a very limited set of schedulers (the so-called *partial-information policies*).

Therefore we focus on the question of model checking but, rather than considering all schedulers to calculate maximum and minimum probabilities, we decide to consider the model checking problem under the subset of *distributed* schedulers that are obtained by composing *local* schedulers (i.e. the schedulers of each component). Notice that the "almighty" scheduler of the example would not be a valid scheduler in this new setting since the choice of G depends only on

information which is external to (and not observable by) G. Distributed schedulers has been studied in [7] in a synchronous setting and in [3] in an asynchronous setting, and are related to the partial-information policies of [6].

The contribution of this paper is to show that the quantitative model checking problem, including pCTL and LTL model checking, restricted to distributed schedulers (called *schedulers for modules* in the rest of the paper) is undecidable. We prove that there is no algorithm to compute the maximum probability of reaching a set of states using only distributed schedulers. Moreover, we show that it is not even possible to compute an approximation of such a value within a given bound. We focus our proof on the synchronous setting of [7] but later show that it also applies to [3] and [6]. Since the model considered here can be encoded in the input language of model checkers such as PRISM or Rapture, our result equally applies to this setting. In other words, the probabilities usually returned by these model checkers overapproximate the ones possible in the more realistic interpretation of distributed schedulers, and there is no way to compute or approximate these values. The proof of undecidability is based on a result of [12] stating that it is undecidable to find an approximation to the maximum probability of accepting a word in a probabilistic finite-state automaton.

Organization of the paper. Next section recalls the model of [7] appropriately simplified to meet our needs. Section 3 presents the tools we use to prove undecidability, including the result of [12] and its relation to our setting. The main proof of undecidability is given in Sec. 4 together with some lemmas which are interesting on their own. In Sec. 5 we also show that our result extends to the models of [3] and [6], discuss other related works. Sec. 6 concludes the paper.

2 Modules and Schedulers

In this section we recall the model of [7] which gives the formal framework to prove our undecidability result. [7] introduces *(probabilistic) modules* to describe open probabilistic systems. Modules can be composed forming more complex modules (though we will not focus on this). On the other hand, modules are built out of *atoms*. Each atom groups the behavior that needs to be scheduled together. Hence, modules can be used to model a distributed system, and atoms can be used to model components in these distributed systems.

Each atom controls a set of variables in an exclusive manner and is allowed to read a set of variables that it may not control. That is, each variable can be modified at any time by only one atom but read by many. The change of values of variables is done randomly according to *moves* and they can take place whenever indicated by a *transition*. Thus an atom is a set of transitions that can only read the variables that the atom may read and can only change (according to some move) the variables that the atom controls. In what follows we give the formalization of these concepts.

Definition 1 (States and moves). *Let X be a set of variables. An X-*state *s is a function mapping each variable in X to a value. An X-*move *a is a probability distribution on X-states. Given scalars $\{\delta_i\}$ and X-moves $\{a_i\}$ such that*

$\sum_i \delta_i = 1$ *and* $\delta_i \geq 0$*, we write* $\sum_i \delta_i a_i$ *for the* X*-move resulting from the convex combination, i.e.,* $\left(\sum_i \delta_i a_i\right)(s) = \sum_i \delta_i a_i(s)$.

In the rest of the paper, we assume that the set of variables is finite as well as the set of their possible values.

Definition 2 (Transitions and convex closures). *Let* X *and* Y *be two sets of variables. A* probabilistic transition (s, a) *from* X *to* Y *consists of an* X*-state* s *and a* Y*-move* a*. Given a set* S *of transitions, the convex closure of* S *(denoted by* $\text{ConvexClosure}(S)$*) is the least set containing all the transitions* $(s, \sum_i \delta_i a_i)$ *for all* $(s, a_i) \in S$ *and* δ_i *as in Def. 1.*

Definition 3 (Atoms). *A* probabilistic X-atom A *consists of a set* $\text{read}_X(A) \subseteq X$ *of* read *variables, a set* $\text{ctr}_X(A) \subseteq X$ *of* controlled *variables and a finite set* $\text{Transitions}(A)$ *of transitions from* $\text{read}_X(A)$ *to* $\text{ctr}_X(A)$.

Atoms in [7] have two sets of transitions: one like ours and another one for initialization. In order to simplify the model, we dropped this second set and consider a unique initial state provided by the module.

Since our result does not require a framework as general as the original, we exclude external and private variables out of our definitions. It is easy to see that the definition of modules below agree with that of [7] when restricted to have only interface variables.

Definition 4 (Modules). *A* probabilistic X-module P *has an initial state and a finite set* $\text{Atoms}(P)$*. We write* $\text{Var}(P)$ *for* X*. The initial state* $\text{Init}(P)$ *is a* $\text{Var}(P)$*-state.* $\text{Atoms}(P)$ *is a finite set of* $\text{Var}(P)$*-atoms such that (1)* $\text{Var}(P) = \bigcup_{A \in \text{Atoms}(P)} \text{ctr}_X(A)$ *and (2)* $\forall A, A' \in \text{Atoms}(P) \bullet \text{ctr}_X(A) \cap \text{ctr}_X(A') = \emptyset$.

The semantics of a deterministic probabilistic system (i.e., Markov chains) is given by a probability distribution on traces (called *bundle* in this context). Nondeterministic probabilistic models –such as modules– exhibit different probabilistic behavior depending on how nondeterministic choices are resolved. Hence, the semantics of a module is given by a *set* of bundles. Each bundle of this set responds to a different way of resolving nondeterminism. The resolution of nondeterminism is done by a *scheduler*. A scheduler is a function that maps traces to distribution on possible moves. Such moves are convex combinations of the available moves at the end of the trace.

Definition 5 (Traces and bundles). *Let* n *be a positive integer. An* X*-trace* σ *of length* n *is a sequence of* X*-states with* n *elements. We write* $\sigma(i)$ *for the* i*-th element of* σ *and* $\text{last}(\sigma)$ *for the last element of* σ*. In addition, we write* $\text{len}(\sigma)$ *for the length of the sequence, and* $\sigma{\downarrow_n}$ *(if* $n \leq \text{len}(\sigma)$*) for the* n-th prefix*, i.e., the sequence of length* n *in which* $(\sigma{\downarrow_n})(i) = \sigma(i)$ *for all* $1 \leq i \leq n$*. We denote the* prefix order *by* $\sigma \sqsubseteq \sigma'$ *if* $\sigma = \sigma'{\downarrow_{\text{len}(\sigma)}}$.

An X*-bundle of length* n *is a probability distribution on* X*-traces of length* n*. The unique* X*-bundle of length* 1*, which assigns the probability* 1 *to the trace consisting only of the initial state is called the* initial bundle.

Definition 6 (Schedulers). *Let X and Y be two sets of variables, and s a starting Y-state. A* scheduler *η from X to Y is a function mapping every X-trace to a probability distribution on Y-states. If η is a scheduler from X to X, then the 1-outcome of η is the bundle $\mathbf{b}_1$ assigning 1 to the trace s. In addition, for all positive integers $i > 1$, the i-outcome of η is an inductively defined $X-$bundle $\mathbf{b}_i$ of length i: the bundle $\mathbf{b}_i$ is the extension of the bundle $\mathbf{b}_{i-1}$ such that $\mathbf{b}_i(\sigma) = \mathbf{b}_{i-1}(\sigma{\downarrow_{i-1}}) \cdot (\eta(\sigma{\downarrow_{i-1}}))(\sigma(i))$ for all $X-$traces of length i. We collect the set of i-outcomes of η ($i \geq 1$) in the set* $\mathrm{Outcome}(\eta)$ *of $X-$bundles. To simplify notation, we write* $\mathrm{Outcome}(\eta)(\sigma)$ *for* $\mathbf{b}_{\mathrm{len}(\sigma)}(\sigma)$.

Schedulers are the machinery to resolve nondeterminism. They do so by assigning probabilities to the different available moves at each point of an execution (i.e., a trace). In our framework, this is done by choosing a move which is a convex combination of the available moves. A scheduler can be *deterministic* (or *non-probabilistic*) in the sense that it assigns probability 1 to a single available move. In other words, a deterministic scheduler chooses a single move from the available ones at each point of the execution. In particular, we are interested on the scheduling within a single component, that is, within an atom.

Definition 7 (Schedulers for an atom). *Consider a probabilistic $X-$atom A. The set* $\mathrm{atom}\Sigma(A)$ *of* atom schedulers *for A contains all schedulers η from* $\mathrm{read}_X(A)$ *to* $\mathrm{ctr}_X(A)$ *such that* $(\sigma(n), \eta(\sigma)) \in \mathrm{ConvexClosure}(\mathrm{Transitions}(A))$ *for all* $\mathrm{read}_X(A)$*-traces σ of length $n \geq 1$. Let* $\mathrm{atom}\Sigma^d(A)$ *be set of deterministic schedulers for A, i.e., the subset of* $\mathrm{atom}\Sigma(A)$ *such that* $(\sigma(n), \eta(\sigma)) \in \mathrm{Transitions}(A)$.

Schedulers for atoms can observe all possible executions and all the (observed) state space. On the contrary, we are *not* interested in any arbitrary scheduler for a module. In a distributed setting each component schedules its own moves disregarding any behavior that does not affect its own state space. Hence, a global scheduler can only be obtained by the combination of the schedulers of each component. Similarly, we consider that a scheduler for a module only makes sense if it comes from the composition of schedulers of each of its atoms. Schedulers for a module are obtained by taking the product of schedulers for atoms as defined in the following.

Definition 8 (Projection and Product). *Let X and $X' \subseteq X$ be two sets of variables. The X'-projection of an X-state s is the X'-state $s[X']$ such that $(s[X'])(x) = s(x)$ for all variables $x \in X'$. The X'-projection of a trace σ is an X'-trace $\sigma[X']$ in which $\sigma[X'](i) = \sigma(i)[X']$ for $1 \leq i \leq \mathrm{len}(\sigma)$.*

Let X_1 and X_2 be two disjoint sets of variables. The product *of an X_1-state s_1 and an X_2-state s_2 is the $X_1 \cup X_2$-state $s_1 \times s_2$ such that $(s_1 \times s_2)(x_1) = s_1(x_1)$ for all $x_1 \in X_1$ and $(s_1 \times s_2)(x_2) = s_2(x_2)$ for all $x_2 \in X_2$. The* product *of an X_1-move a_1 and an X_2-move a_2 is the $X_1 \cup X_2$-move $a_1 \times a_2$ such that $(a_1 \times a_2)(s) = a_1(s[X_1]).a_2(s[X_2])$.*

Definition 9 (Product of schedulers). *If η_1 is a scheduler from X_1 to Y_1, and η_2 is a scheduler from X_2 to Y_2, such that $Y_1 \cap Y_2 = \emptyset$, then the* product

is the scheduler $\eta_1 \times \eta_2$ *from* $X_1 \cup X_2$ *to* $Y_1 \cup Y_2$ *such that* $(\eta_1 \times \eta_2)(\sigma) = \eta_1(\sigma[X_1]) \times \eta_2(\sigma[X_2])$ *for all* $X_1 \cup X_2$*-traces* σ*. If* Σ_1 *and* Σ_2 *are two sets of schedulers, then* $\Sigma_1 \times \Sigma_2 = \{\eta_1 \times \eta_2 | \eta_1 \in \Sigma_1 \text{ and } \eta_2 \in \Sigma_2\}$.

Since we restrict to modules without external variables, our definition of scheduler for a module is simpler than that of [7]. It is easy to see that the definition of schedulers below agree with that in [7] when restricted to our setting.

Definition 10 (Schedulers for a module). *Consider a probabilistic module* P*. The set* $\mathrm{mod}\Sigma(P)$ *of* module schedulers *for* P *contains all the schedulers from* $\mathrm{Var}(P)$ *to* $\mathrm{Var}(P)$ *having* $\mathrm{Init}(P)$ *as starting state which can be written as a product of schedulers for the atoms, i.e.:* $\mathrm{mod}\Sigma(P) = \{\prod_{A \in \mathrm{Atoms}(P)} \eta_A \mid \eta_A \in \mathrm{atom}\Sigma(A)\}$*. Let* $\mathrm{mod}\Sigma^d(P)$ *be the set of deterministic schedulers for the module, i.e.,* $\mathrm{mod}\Sigma^d(P) = \{\prod_{A \in \mathrm{Atoms}(P)} \eta_A \mid \eta_A \in \mathrm{atom}\Sigma^d(A)\}$.

Each scheduler defines a probability space on the set of infinite traces as stated below.

Definition 11 (Extensions and probability of traces). *For each* finite *trace* σ *of length* n*, we define the* set of extensions $[\sigma]$ *to be set of* infinite *traces such that for every* $\rho \in [\sigma]$*,* $\rho(n') = \sigma(n')$ *for all* $1 \leq n' \leq n$.

Let be P *a probabilistic module whose variables are* $\mathrm{Var}(P)$*, and let* $\eta \in \mathrm{mod}\Sigma(P)$ *be a scheduler for* P*. The probability* $\Pr^\eta([\sigma])$ *of a set of extensions is* $\mathrm{Outcome}(\eta)(\sigma)$*. This probability can be extended in the standard way to the least* σ*-algebra over the set of infinite traces containing the extensions (see [11]).*

Given the setting in the previous definition, one can talk of the probability $\Pr^\eta(\mathrm{reach}(U))$ of reaching the set of states U under η. Such probability is given by $\Pr^\eta(\{\rho \mid \exists n \bullet \rho(n) \in U\})$.

In the following, we define several shorthands and notations that will be convenient for the rest of the paper.

Let $\mathrm{en}_A(s)$ be the set of enabled moves in a Y-state s of an X-atom A with $X \subseteq Y$; that is, the set $\{a \mid (s[\mathrm{read}_X(A)], a) \in \mathrm{Transitions}(A)\}$.

A scheduler η for an X-atom A is a function from $\mathrm{read}_X(A)$-traces to distributions on $\mathrm{ctr}_X(A)$ such that $(\mathrm{last}(\sigma), \eta(\sigma)) \in \mathrm{ConvexClosure}(\mathrm{Transitions}(A))$. That is, $\eta(\sigma) = \sum_{a \in \mathrm{en}_A(s)} \delta_a a$, where $\sum_{a \in \mathrm{en}_A(s)} \delta_a = 1$. Hence, η can be alternatively seen as a function $\eta_f : \mathrm{ctr}_X(A)\text{-moves} \times \mathrm{read}_X(A)\text{-traces} \to [0,1]$, s.t. $\eta_f(a, \sigma) = \delta_a$ for all $a \in \mathrm{en}_A(s)$, and 0 otherwise. Conversely, if $\eta_f(a, \sigma) > 0 \Rightarrow a \in \mathrm{en}_A(\mathrm{last}(\sigma))$ and $\sum_{a \in \mathrm{en}_A(\mathrm{last}(\sigma))} \eta_f(a, \sigma) = 1$, for all trace σ, η_f defines a scheduler η s.t. $\eta(\sigma) = \sum_{a \in \mathrm{en}_A(s)} \eta_f(a, \sigma)\ a$. We will use η and η_f interchangeably according to our convenience.

If η is a scheduler for the module P, we will call η_A to the scheduler for the atom A of P such that $\eta = \eta_A \times \prod_{A' \in \mathrm{Atoms}(P) \setminus \{A\}} \eta_{A'}$. If η is deterministic and $\mathrm{Atoms}(P) = A_1, \ldots, A_n$ we ambiguously denote by $\eta(\sigma)$ the n-tuple of moves $(a_1, \ldots, a_n)$ such that $\eta(\sigma)(a_1, \ldots, a_n) = 1$. Notice that $\eta_{A_i}(\sigma) = a_i = \pi_i(\eta(\sigma))$ for all $1 \leq i \leq n$.

3 The Setting for the Proof of Undecidability

In this section we present the foundations for the proof of undecidability. We recall the result in [12] stating that it is undecidable to find an approximation to the maximum probability of accepting a word in a probabilistic finite-state automaton (PFA), and present a translation of PFAs into probabilistic modules. This setting is used in Sec. 4 to prove that it is not possible to determine the maximum probability of reaching a given set of states in a module. The maximum reachability problem is formally stated as follows:

Definition 12 (Maximum Reachability Problem). *Let P be a module, let U be a set of* $\mathrm{Var}(P)$*-states, and let* $\mathrm{reach}(U) = \{\rho \mid \exists n \bullet \rho(n) \in U\}$ *be the set of all infinite traces that pass through some state in U. The* maximum reachability problem *is to determine* $\sup_{\eta \in \mathrm{mod}_\Sigma(P)} \Pr^\eta(\mathrm{reach}(U))$.

In the following, we assume that any trace that reaches some state in U remains in U with probability 1. It is a standard assumption in reachability analysis of Markov decision processes in general to make target states absorbing (see e.g. [14,1,12]). The assumption in our setting is formally stated as follows.

Assumption 1 (States in U are absorbing). *Given an instance of the maximum reachability problem, we assume that the elements in U are absorbing, in the sense that* $\forall s \in U \bullet \Pr^\eta([\sigma\, s]) = \Pr^\eta(\biguplus_{s' \in U} [\sigma\, s\, s'])$ *for all η.*

The proof presented in the next section is based on the reduction of the probabilistic finite-state automata (PFA) maximum acceptance problem [12] to the maximum reachability problem on a module. Since this problem is undecidable, this reduction implies the undecidability of the maximum reachability problem.

A PFA is a quintuple (Q, Σ, l, q_i, q_f) where Q is a finite set of *states* with $q_i, q_f \in Q$ being the *initial* and *accepting state* respectively, Σ is the *input alphabet*, and $l : \Sigma \times Q \to (Q \to [0,1])$ is the *transition function* s.t. $l(\alpha, q)$ is a distribution for all $\alpha \in \Sigma$ and $q \in Q$. Notice that l is a total function. As in [12], we assume that q_f is absorbing, i.e. $l(\alpha, q_f)(q_f) = 1$ for all $\alpha \in \Sigma$.

In the following, we present the translation of PFA into modules and directly define the probability of accepting a word in the translated model. We do so to avoid introducing a probabilistic theory on traces for a slightly different setting.

A PFA is encoded in a module P_{PFA} with two variables and two atoms. Variable *st_pfa* takes values in Q recording the current state in the PFA. Variable *symbol* takes values in $\Sigma \cup \{\mathit{initial}\}$ and is used to indicate the next input issued to the PFA. In particular, *initial* is introduced for technical matters and only used in the first transition of the module to indicate that no selection has being issued yet. Atom A encodes the transition function l. Therefore it can read both variables, but can only control *st_pfa*. It is atom B the one that controls variable *symbol* and the one that introduces the nondeterminism in the selection of the input. Notice that A is completely deterministic (in the sense that, at every state, the value of *symbol* uniquely determines the next transition to execute). Since B takes the role of the environment selecting inputs, it cannot read (nor

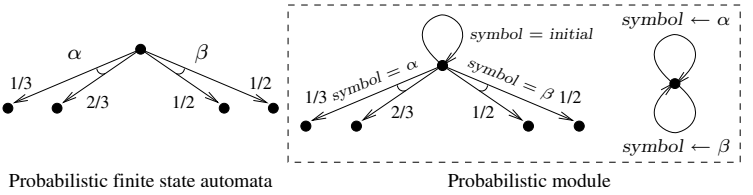


Fig. 2. From PFA to probabilistic modules

control) variable *st_pfa*. Hence, a word w over Σ is equivalent to the deterministic scheduler for B that chooses the symbols in the word.

The definition of P_{PFA} is formalized in the following.

Definition 13 (P_{PFA} and the probability of accepting a word). *Let* $X = \{st\_pfa, symbol\}$. *Let* A *be an* X*-atom with* $\text{read}_X(A) = X$ *and* $\text{ctr}_X(A) = \{st\_pfa\}$, *such that (1)* $\text{en}(s) = \{a_{l,s}\}$ *for all* s *such that* $s(symbol) \neq initial$, *where* $a_{l,s}(s') = l(s(symbol), s(st\_pfa))(s'(st\_pfa))$ *for all* $\{st\_pfa\}$*-state* s', *and (2)* $\text{en}(s) = \{a_s\}$ *for all* s *such that* $s(symbol) = initial$, *where* $a_s(s) = 1$. *Let* B *be an* X*-atom with* $\text{read}_X(B) = \text{ctr}_X(B) = \{symbol\}$, *and for all state* s *and symbol* α *it contains transition* (s, a_α), *where* $a_\alpha(symbol = \alpha) = 1$. P_{PFA} *is defined as the module containing atoms* A *and* B *above and having initial state* s_i *such that* $s_i(st\_pfa) = q_i$ *and* $s_i(symbol) = initial$.

Let $U = \{s \mid s(st\_pfa) = q_f\}$ *be the set of accepting states. Then, the probability* $\Pr(\text{accepting } w)$ *of accepting an infinite word* $w = w_1 w_2 \cdots$ *of symbols from* Σ *is* $\Pr^{\eta_A \times \eta_B}(\text{reach}(U))$, *where* $\eta_B(\sigma) = a_{w_{\text{len}(\sigma)-1}}$ *(if* $\text{len}(\sigma) > 1$*),* $\eta_B(s) = a_s$ *and* η_A *is the only possible deterministic scheduler for* A.

Atom A is deterministic, since it has exactly one enabled move at every state. Hence there exists only one possible scheduler for A (the scheduler choosing the only possible move). In addition, it is also worth noting that, although we are dealing with infinite words, our criterion for acceptance is to *pass through* the accepting state using the word (i.e., a word is accepted iff a finite prefix reaches the accepting state). Figure 2 shows a simple PFA and its corresponding probabilistic module. In this figure, $symbol = \alpha$ indicates that the transition needs the value of the variable *symbol* to be α. In addition, $symbol \leftarrow \alpha$ indicates that the transition sets the value of the variable *symbol* to α.

Stated in terms of Def. 13, Corollary 3.4 in [12] states the following:

Lemma 1 (Corollary 3.4 in [12]). *For any fixed* $0 < \epsilon < 1$, *the following problem is undecidable: Given a module* P_{PFA} *as in Def. 13 such that either*

1. P_{PFA} *accepts some word with probability greater than* $1 - \epsilon$, *or*
2. P_{PFA} *accepts no word with probability greater then* ϵ;

decide whether case 1 holds.

[12] points out that, as a consequence of Lemma 1, the approximation of the maximum acceptance probability is also undecidable. This statement is formalized in the following corollary.

Corollary 1 (Approximation of the maximum acceptance probability is undecidable). *Given P_{PFA} as in Def. 13 and $\delta > 0$, the following problem is undecidable: find r such that $|r - \sup_w \Pr(\textit{accept } w)| < \delta$.*

4 Undecidability of the Reachability Problem for Modules

In this section, we prove the undecidability of the maximum reachability problem. Recall that the maximum reachability problem is to find the supremum over the set of all the schedulers for a given module. First of all, we show that infinite words can be seen as deterministic schedulers (Lemma 2). So, the problem of finding the supremum over the set of words is equivalent to the problem of finding the supremum over the set of *deterministic* schedulers. Next, we prove that the supremum quantifying over deterministic schedulers equals the supremum quantifying over all schedulers (Lemma 4) using the fact that, given a scheduler and a number N, a deterministic scheduler can be found which yields a larger probability until the N-th step (Lemma 3).

In the following, we prove not only the undecidability of the maximum reachability problem on modules, but also that the value of the maximum reachability probability cannot be approximated, i.e. given a certain threshold δ, there is no algorithm returning r such that $|r - \sup_{\eta \in \mathrm{mod}_\Sigma(P)} \Pr^\eta(\mathrm{reach}(U))| < \delta$.

The following lemma states that each word in the PFA can be seen as a deterministic scheduler in P_{PFA} and vice versa.

Lemma 2 (Words and schedulers). *Given P_{PFA} as in Def. 13, each word w corresponds to a deterministic scheduler η and vice versa, in the sense that $\Pr(\textit{accepting } w) = \Pr^\eta(\mathrm{reach}(U))$.*

Proof. By definition, $\Pr(\text{accepting } w) = \Pr^{\eta_A \times \eta_B}(\mathrm{reach}(U))$, with η_A and η_B as in Def. 13.

Conversely, let η be a deterministic scheduler for P_{PFA}. Then $\eta = \eta_A \times \eta_B$ for some η_A (which is unique) and η_B. Note that, for any $n > 0$, there is exactly one $\{symbol\}$-trace σ_n having probability greater than 0 and $\mathrm{len}(\sigma_n) = n$ which is defined by η_B. This is due to the fact that B has no probabilistic transitions and A cannot change the variable *symbol*. Then, take $w = w_1 \cdot w_2 \cdots$ to be the word defined by $w_n = \mathrm{last}(\sigma_{n+1})$. ($\sigma_1$ is ignored since variable *symbol* has the value *initial* in the first state.) □

As a consequence of Lemma 2 and Corollary 1 the computation of the maximum reachability probability restricted to deterministic schedulers –i.e. the computation of $\sup_{\eta \in \mathrm{mod}_{\Sigma^d}(P)} \Pr^\eta(\mathrm{reach}(U))$– is an undecidable problem in general since it is undecidable for the particular case of modules obtained from PFA as in Def. 13. However, this fact does not guarantee that the problem is also undecidable when all module schedulers (not only deterministic ones) are considered (in fact, the problem *is* decidable for arbitrary global schedulers [1,5]). Our main contribution is to show that the problem is undecidable even if schedulers are not restricted to be deterministic.

The following lemma states that given a scheduler and a bound N, there is a deterministic scheduler that yields a larger probability of reaching U within the first N steps.

Lemma 3. *Given a scheduler η and $N \in \mathbb{N}$ there exists a deterministic scheduler η_d such that $\Pr^{\eta_d}(\text{reach}_N(U)) \geq \Pr^{\eta}(\text{reach}_N(U))$, where $\Pr^{\eta}(\text{reach}_N(U))$ denotes the probability of reaching U before the N-th step.*

Proof. Given an atom A^*, a $\text{read}_X(A^*)$-trace σ^* such that $\text{len}(\sigma^*) \leq N$ and a scheduler $\eta = \prod_{A \in \text{Atoms}(P)\setminus\{A^*\}} \eta_A \times \eta_{A^*}$, we find a scheduler $\det(\eta, \sigma^*)$ such that $\det(\eta, \sigma^*)$ coincides with η except for the choice corresponding to the trace σ^* in the atom A^*, in which $\det(\eta, \sigma^*)$ deterministically chooses a single action. Formally, $\det(\eta, \sigma^*)$ can be expressed in terms of η as follows: $\det(\eta, \sigma^*) = \prod_{A \in \text{Atoms}(P)\setminus\{A^*\}} \eta_A \times \eta'_{A^*}$, with $\eta'_{A^*}(\sigma^*, a^*) = 1$ for some a^* and $\eta'_{A^*}(\sigma, a) = \eta_{A^*}(\sigma, a)$ for all $\sigma \neq \sigma^*$. In addition, in the construction of $\det(\eta, \sigma^*)$ we choose a^* such that $\Pr^{\det(\eta,\sigma^*)}(\text{reach}_N(U)) \geq \Pr^{\eta}(\text{reach}_N(U))$. The core of the proof is to find such an a^*. Once obtained $\det(\eta, \sigma^*)$ for a trace σ^*, the final deterministic scheduler is calculated by repeating this process for all the local traces with length less than or equal to N.

In the following, let $k = \text{len}(\sigma^*)$ and define $r_N = \{\sigma \mid \text{len}(\sigma) = N \wedge \sigma(N) \in U\}$, $r_{N,\sigma^*} = r_N \cap \{\sigma \mid \sigma{\downarrow_k}[\text{read}_X(A^*)] = \sigma^*\}$ and $r_{N,\neg\sigma^*} = r_N \setminus r_{N,\sigma^*}$.

Note that, because of Assumption 1, $\Pr^{\eta}(\text{reach}_N(U)) = \Pr^{\eta}(\biguplus_{\sigma \in r_N} [\sigma])$, since $\Pr^{\eta}(\biguplus_{s'_i \in U} [\sigma s s'_1 \cdots s'_{N-(\text{len}(\sigma)+1)}]) = \Pr^{\eta}([\sigma s])$ for all $s \in U$.

Now, we start the calculations to find a^*.

$$\begin{aligned}
\Pr^{\eta}(\text{reach}_N(U)) &= \Pr^{\eta}(\biguplus_{\sigma \in r_N} [\sigma]) && \{\text{Explanation above}\}\\
&= \textstyle\sum_{\sigma \in r_N} \Pr^{\eta}([\sigma]) && \{\Pr \text{ is a measure}\}\\
&= \textstyle\sum_{\sigma \in r_{N,\sigma^*}} \Pr^{\eta}([\sigma]) + \sum_{\sigma \in r_{N,\neg\sigma^*}} \Pr^{\eta}([\sigma]) && \{\text{Commutativity}\}
\end{aligned}$$

Next, we examine the first summand. In the following calculation, let $P_{\sigma,i,A} = \sum_{a \in \text{en}_A(\sigma(i))} \eta_A(\sigma{\downarrow_i}[\text{read}_X(A)], a) \;\; a(\sigma(i+1))$.

$$\begin{aligned}
&\textstyle\sum_{\sigma \in r_{N,\sigma^*}} \Pr^{\eta}([\sigma])\\
=\;& \{\text{Definition 11}\}\\
&\textstyle\sum_{\sigma \in r_{N,\sigma^*}} \prod_{i=1}^{N-1} \prod_{A \in \text{Atoms}(P)} \sum_{a \in \text{en}_A(\sigma(i))} \eta_A(\sigma{\downarrow_i}[\text{read}_X(A)], a) \;\; a(\sigma(i+1))\\
=\;& \{\text{Arithmetics}\}\\
&\textstyle\sum_{\sigma \in r_{N,\sigma^*}} \Pr^{\eta}([\sigma{\downarrow_k}])\\
&\qquad \big(\textstyle\sum_{a \in \text{en}_{A^*}(\sigma(k))} \eta_{A^*}(\sigma{\downarrow_k}[\text{read}_X(A^*)], a) \;\; a(\sigma(k+1))\big)\\
&\qquad \textstyle\prod_{A \in \text{Atoms}(P)\setminus\{A^*\}} P_{\sigma,k,A}\\
&\qquad \textstyle\prod_{i=k+1}^{N-1} \prod_{A \in \text{Atoms}(P)} P_{\sigma,i,A}\\
=\;& \{\sigma{\downarrow_k}[\text{read}_X(A^*)] = \sigma^* \text{ (since } \sigma \in r_{N,\sigma^*}\text{), arithmetics}\}\\
&\textstyle\sum_{a \in \text{en}_{A^*}(\sigma^*(k))} \eta_{A^*}(\sigma^*, a)\\
&\qquad \textstyle\sum_{\sigma \in r_{N,\sigma^*}} \Pr^{\eta}([\sigma{\downarrow_k}]) \; a(\sigma(k+1))\\
&\qquad \textstyle\prod_{A \in \text{Atoms}(P)\setminus\{A^*\}} P_{\sigma,k,A}\\
&\qquad \textstyle\prod_{i=k+1}^{N-1} \prod_{A \in \text{Atoms}(P)} P_{\sigma,i,A}
\end{aligned}$$

$$\text{Let} \quad a^* = \arg\max_{a \in \mathrm{en}_{A^*}(\sigma(k))} \sum_{\sigma \in r_{N,\sigma^*}} \Pr^\eta([\sigma\downarrow_k])\; a(\sigma(k+1)) \prod_{A \in \mathrm{Atoms}(P)\setminus\{A^*\}} P_{\sigma,k,A} \prod_{i=k+1}^{N-1} \prod_{A \in \mathrm{Atoms}(P)} P_{\sigma,i,A}\,.$$

Then, since $\sum_{a \in \mathrm{en}_{A^*}(\sigma(k))} \eta_{A^*}(\sigma^*, a) = 1$, we have

$$\begin{aligned}\sum_{\sigma \in r_{N,\sigma^*}} \Pr^\eta([\sigma]) &\le \sum_{\sigma \in r_{N,\sigma^*}} \Pr^\eta([\sigma\downarrow_k])\; a^*(\sigma(k+1)) \prod_{A \in \mathrm{Atoms}(P)\setminus\{A^*\}} P_{\sigma,i,A} \prod_{i=k+1}^{N-1} \prod_{A \in \mathrm{Atoms}(P)} P_{\sigma,i,A} \\ &= \sum_{\sigma \in r_{N,\sigma^*}} \Pr^{\eta'}([\sigma])\,,\end{aligned}$$

where η' is the scheduler that coincides with η except for trace σ^*, in which the scheduler for the atom A^* chooses a^*.

Since this change does not affect the extensions in $r_{N,\neg\sigma^*}$, we have that $\Pr^{\eta'}(r_N(U)) \ge \Pr^\eta(r_N(U))$. Thus, we define $\det(\eta, \sigma^*) = \eta'$.

Given a sequence of local traces $\sigma_1 \cdots \sigma_n$ (possibly belonging to different atoms), we extend the definition of det in order to handle finite sequences of traces as follows: $\det(\eta, \sigma_1 \cdots \sigma_n) = \det(\det(\eta, \sigma_n), \sigma_1 \cdots \sigma_{n-1})$.

Now, we can define a scheduler η^N being deterministic "until the N-th step" by considering the sequence $\sigma_1 \cdots \sigma_M$ comprising all local traces whose length is less or equal than N and computing $\det(\eta, \sigma_1 \cdots \sigma_M)$.

Since the choices after the N-th step do not affect the value of $\Pr(\mathrm{reach}_N(U))$, we construct the desired scheduler by taking η^N and modifying it to deterministically choose any move after the N-th step. □

Using the previous lemma, we prove that the maximum probability of reaching U is the same regardless whether it is quantified over all schedulers or only over deterministic schedulers.

Lemma 4. $\sup_{\eta \in \mathrm{mod}_{\Sigma^d}(P)} \Pr^\eta(\mathrm{reach}(U)) = \sup_{\eta \in \mathrm{mod}_{\Sigma}(P)} \Pr^\eta(\mathrm{reach}(U))$

Proof. Let $r = \sup_{\eta \in \mathrm{mod}_{\Sigma}(P)} \Pr^\eta(\mathrm{reach}(U))$. We prove that for every ϵ, there exists a deterministic scheduler η_ϵ such that $r - \Pr^{\eta_\epsilon}(\mathrm{reach}(U)) < \epsilon$, thus proving the lemma.

Given $\epsilon > 0$, there exists a scheduler η such that $r - \Pr^\eta(\mathrm{reach}(U)) < \epsilon/2$.

Note that we can write $\mathrm{reach}(U)$ as

$$\biguplus_{\{n \in \mathbb{N}_0\}} \biguplus_{\{\sigma' \mid \forall i \bullet \sigma(i) \notin U \wedge \mathrm{len}(\sigma') = n\}} \biguplus_{\{s \in U\}} [\sigma' \cdot s]\,.$$

Then, since Pr is a measure

$$\Pr^\eta(\mathrm{reach}(U)) = \sum_{\{n \in \mathbb{N}_0\}} \sum_{\{\sigma' \mid \forall i \bullet \sigma(i) \notin U \wedge \mathrm{len}(\sigma') = n\}} \sum_{\{s \in U\}} \Pr^\eta([\sigma' \cdot s])\,.$$

So, there exists N such that

$$\Pr^\eta(\mathrm{reach}(U)) - \sum_{n=0}^{N} \sum_{\{\sigma' \mid \forall i \bullet \sigma(i) \notin U \wedge \mathrm{len}(\sigma') = n\}} \sum_{\{s \in U\}} \Pr^\eta([\sigma' \cdot s]) < \epsilon/2\,.$$

By virtue of Lemma 3, we know that there exists a deterministic scheduler η_d such that $\Pr^{\eta_d}(\text{reach}_N(U)) \geq \Pr^{\eta}(\text{reach}_N(U))$. Then, $\Pr^{\eta_d}(\text{reach}(U)) \geq \Pr^{\eta}(\text{reach}(U)) - \epsilon/2$. This result yields,

$$\begin{aligned} r - \Pr^{\eta_d}(\text{reach}(U)) &= r - \Pr^{\eta}(\text{reach}(U)) + \Pr^{\eta}(\text{reach}(U)) - \Pr^{\eta_d}(\text{reach}(U)) \\ &< \epsilon/2 + \epsilon/2 = \epsilon\,. \end{aligned}$$

□

Though Lemma 4 is the basis for our undecidability result, it has a value of its own: it states that, for any probabilistic module P and reachability target U, it suffices to consider only deterministic schedulers to calculate the maximum probability of reaching some state in U. Lemma 4 yields to our main result:

Theorem 1 (Approximation of the maximum reachability problem is undecidable). *Given a probabilistic module P, a set U of states and $\delta > 0$, there is no algorithm that computes r such that $|r - \sup_{\eta \in \text{mod}\Sigma(P)} \Pr^{\eta}(\text{reach}(U))| < \delta$.*

Proof. Suppose, towards a contradiction, that the problem is decidable. Take an instance of P_{PFA} as in Def. 13. Then, using Lemmas 4 and 2, we can compute r such that

$$\begin{aligned} \delta &> |r - \sup_{\eta \in \text{mod}\Sigma(P)} \Pr^{\eta}(\text{reach}(U))| = |r - \sup_{\eta \in \text{mod}\Sigma^d(P)} \Pr^{\eta}(\text{reach}(U))| \\ &= |r - \sup_{w} \Pr(\text{accept } w)| \end{aligned}$$

thus contradicting Corollary 1. □

Often reachability properties are only of interest if they are compared to a probability value (e.g. the maximum probability of an error is smaller than 0.01). This kind of problems are also undecidable. If this were not the case, a procedure to calculate an approximation to the maximum reachability probability can be easily constructed using bisection (see, e.g., [13]). This result is formally stated in the following corollary.

Corollary 2. *Let $\star$ denote some operator in $\{\leq, \geq, <, >, =\}$. There is no algorithm that returns yes if $\sup_{\eta} \Pr^{\eta}(\text{reach}(U)) \star q$ or returns no, otherwise, for a given module P and number q.*

Moreover, there exists no algorithm that, given a module P, a number q and a threshold ϵ returns yes if $r \star q$ for some r such that $|r - \sup_{\eta} \Pr^{\eta}(\text{reach}(U))| < \epsilon$ or returns no, otherwise.

5 Impact and Related Work

Undecidability is frequent in problems involving control and partial information (e.g. [18]). Since a scheduler can be seen as a controller which enables appropriate moves, control is closely related to scheduling. This fact gave us a clue about the result presented in this paper. In [18], a finite state automaton can execute an action only if a set of infinite-state controllers allows it. The state of a controller (and, hence, the actions it allows to execute) is updated each time an action

observed by the controller happens. Although schedulers can be seen as infinite-state automata —since the language obtained by taking the (distributions on) moves prescribed by the scheduler is not restricted— we could not prove our result using the results in [18] by simply taking the controllers to be schedulers. Moreover, global schedulers are also infinite state automata, and the problem for these schedulers is decidable (see [5]). These facts suggest that the tractability of this problem is likely to vary from a formalism to another, although the formalisms under consideration may seem to be similar at first sight.

Unfortunately, our result also holds for Switched Probabilistic Input/Output Automata (Switched PIOA) [3,2] and for the schema of partial information presented in [6].

In the Switched PIOA formalism, the different components have input and output local schedulers, and a token is used in order to decide the next component to execute. The interleaving between different components is not resolved by the schedulers, since the way in which the token is passed is specified by the components. If a component has the token, its local output scheduler chooses a transition from a generative structure. Otherwise, its local input scheduler chooses a transition from a reactive structure, thus "reacting" to actions performed by the other components. (For definitions of reactive and generative structures see e.g. [8,16].)

The probabilistic finite-state automata in [12] can be seen as components having only reactive structures. In addition, the input scheduler for these components is uniquely determined, since each symbol uniquely determines the probability distribution for the next state. This fact allows to prove undecidability by composing the PFA (seen as a component of the Switched PIOA) with an automaton having only one state which chooses the next action to perform using generative structures (which are Dirac distributions for the sole state of the component). This component has the token during all the course of the execution. So, while this latter component chooses any action, the former component reacts to this choice as the PFA would do. Hence, these two components can simulate a probabilistic module as the one described in Def. 13. Note that we are working with a very strict subset of the Switched PIOA: there are no nondeterministic choices for the inputs, all generative structures are Dirac distributions and the token is owned by the same component during all the course of the execution. It is also worth noting that schedulers as presented in [3,2] are always deterministic because they can choose any transition in a generative or reactive structure, but they cannot choose convex combinations of these transitions. Thus, Lemma 4, and hence Lemma 3, are not needed for Switched PIOA as presented originally, and undecidability can be proved almost directly using Corollary 3.4 in [12]. Our result indicates that the problem remains undecidable even if we extend Switched PIOA with nondeterministic schedulers.

Though composition in [6] is not an issue, it presents schedulers for Markov decision processes which can observe partial portions of the states. The goal is to obtain better bounds for the probability of temporal properties. Markov decision process are defined using a set of actions in such a way that each pair

(s, a) determines the probability distribution for the next state. The schedulers are restricted as follows: given an equivalence relation $\sim$ over the set of states ($s \sim s'$ denoting that the scheduler cannot distinguish s and s'), the relation $\sigma \sim \sigma'$ over traces is defined to hold iff $\text{len}(\sigma) = \text{len}(\sigma')$ and for all i, $\sigma(i) \sim \sigma'(i)$. Then, a partial-information scheduler is required to satisfy $\eta(\sigma, a) = \eta(\sigma', a)$ if $\sigma \sim \sigma'$. Note that, by taking $\sim$ such that $s \sim s'$ for all s, the scheduler must decide the next action to perform based solely on the amount of actions chosen before. Then, using such a relation $\sim$, a scheduler in [6] is equivalent to a scheduler for the atom B as Def. 13, and hence the problem of finding the maximum reachability probability is equivalent to the problem of finding the maximum reachability probability for a module as in Def. 13.

We remark that [6] defines a model checking algorithm, but it calculates the supremum corresponding to *Markovian* partial-information policies, i.e., to the subset of partial-information policies restricted to choose (distributions on) actions by reading only the (corresponding portion of the) current state rather than the full past history. Quantitative model checking on MDP was originally introduced in [1,5] but for arbitrary schedulers. In particular, [5] proves that the maximum reachability problem under *arbitrary global* schedulers has an equivalent solution under *deterministic global* schedulers (in fact, they are also *Markovian* in the sense that only depend on the last state and not of the full trace, see Theorem 3.5 in [5]). Though this result is similar to Lemma 4, the proof of [5] has no connection to ours. In fact, the construction of the deterministic scheduler in [5] is also the proof that the problem for the general case is decidable.

6 Conclusion

We have argued that usual quantitative model checkers yields overestimations of the extremum probabilities in distributed programs and proposed to address model checking under the restriction of schedulers that are compatible with the expected behaviour of distributed systems. We showed that it is undecidable to compute the maximum probability of reaching a state when restricted to distributed schedulers, hence making the proposal unfeasible in its generality. Morever, we showed that such value cannot even be aproximated.

On proving undecidability, we needed to prove additional lemmas. In particular, we believe that the result of Lemma 4 has to be remarked, but mostly, that its proof technique, including the constructive proof of Lemma 3, is of relevance and can be reused in searching similar results.

The combination of our undecidability result and the NP-hardness result of [6] is not encouraging on seeking algorithms for model checking under distributed or partial-information schedulers. Yet, the observation that arbitrary schedulers yield overestimations of probability values remains valid. The question then is whether it is possible to find a proper subset of schedulers (surely including all distributed schedulers) that yields a tighter approximation of extremum probabilities while keeping tractability of the model checking problem.

Another question is to which extent the calculation (or approximation) of the *minimum* probability of reaching a state is also undecidable. Though this

is the dual problem to that in Def. 12, we could not obtain a straightforward dualization of our proof.

Acknowledgement. We would like to thank Peter Niebert who has been an excelent sparring during the development of these ideas, and Holger Hermanns for his most valuable feedback.

References

1. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 1026, pp. 288–299. Springer, Heidelberg (1995)
2. Cheung, L.: Reconciling Nondeterministic and Probabilistic Choices. PhD thesis, Radboud Universiteit Nijmegen (2006)
3. Cheung, L., Lynch, N., Segala, R., Vaandrager, F.W.: Switched Probabilistic PIOA: Parallel composition via distributed scheduling. Theoretical Computer Science 365(1-2), 83–108 (2006)
4. Ciesinski, F., Baier, C.: Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In: Proc. of QEST'06, pp. 131–132. IEEE Computer Society Press, Los Alamitos (2006)
5. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University (1997)
6. de Alfaro, L.: The verification of probabilistic systems under memoryless partial-information policies is hard. In: Proc. of PROBMIV 99. Tech. Rep. CSR-99-8, pp. 19–32. Univ. of Birmingham (1999)
7. de Alfaro, L., Henzinger, T.A., Jhala, R.: Compositional methods for probabilistic systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 351–365. Springer, Heidelberg (2001)
8. van Glabbeek, R.J., Smolka, S.A., Steffen, B.: Reactive, generative, and stratified models of probabilistic processes. Information and Computation 121, 59–80 (1995)
9. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006 and ETAPS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
10. Jeannet, B., D'Argenio, P.R., Larsen, K.G.: Rapture: A tool for verifying Markov Decision Processes. In: I. Cerna, editor, Tools Day'02, Brno, Czech Republic, Technical Report. Faculty of Informatics, Masaryk University Brno (2002)
11. Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov Chains. Van Nostrand Company (1966)
12. Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and related stochastic optimization problems. Artif. Intell. 147(1-2), 5–34 (2003)
13. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd edn. pp. 343–347. Cambridge University Press, Cambridge (1992)
14. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley, Chichester (1994)
15. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, Massachusetts Institute of Technology (1995)

16. Sokolova, A., de Vink, E.P.: Probabilistic automata: system types, parallel composition and comparison. In: Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P., Siegle, M. (eds.) Validation of Stochastic Systems. LNCS, vol. 2925, pp. 1–43. Springer, Heidelberg (2004)
17. Stoelinga, M.: Alea jacta est: Verification of Probabilistic, Real-time and Parametric Systems. PhD thesis, Katholieke Universiteit Nijmegen (2002)
18. Tripakis, S.: Undecidable problems of decentralized observation and control. In: Proc. 40th IEEE Conference on Decision and Control, vol. 5, pp. 4104–4109 (2001)
19. Varacca, D., Nielsen, M.: Probabilistic Petri nets and mazurkiewicz equivalence 2003 (Unpublished draft)

Efficient Detection of Zeno Runs in Timed Automata*

Rodolfo Gómez and Howard Bowman

University of Kent, Computing Laboratory,
CT2 7NF, Canterbury, Kent, United Kingdom
{R.S.Gomez,H.Bowman}@kent.ac.uk

Abstract. Zeno runs, where infinitely many actions occur in finite time, may inadvertently arise in timed automata specifications. Zeno runs may compromise the reliability of formal verification, and few model-checkers provide the means to deal with them: this usually takes the form of liveness checks, which are computationally expensive. As an alternative, we describe here an efficient static analysis to assert absence of Zeno runs on Uppaal networks; this is based on Tripakis's strong non-Zenoness property, and identifies all loops in the automata graphs where Zeno runs may possibly occur. If such unsafe loops are found, we show how to derive an abstract network that over-approximates the loop behaviour. Then, liveness checks may assert absence of Zeno runs in the original network, by exploring the reduced state space of the abstract network. Experiments show that this combined approach may be much more efficient than running liveness checks on the original network.

Keywords: Zeno Runs, Timed Automata, Model-checking, Uppaal.

1 Introduction

Timed automata [1] are often used to specify timed systems, as they provide a graphic notation that is easy to use, and can be automatically verified [2,3,4]. However, specifications may exhibit so-called Zeno runs, which are executions where actions occur infinitely often in a finite period of time. Knowing whether Zeno runs occur increases our confidence on the specification at hand. Zeno runs are unintended (real processes cannot execute infinitely fast); in general, the verification of correctness properties cannot be trusted in specifications where Zeno runs may occur. For example, liveness properties are usually meaningless unless time-divergence is assumed. In addition, Zeno runs may conceal deadlocks and thus compromise safety properties. In most timed automata models, the semantics of urgency allow states (so-called timelocks) where time-divergent runs are no longer possible. Typically, as progress depends on delays, a timelock will have a global effect and make part of the state space unreachable. Hence,

* This research has been supported by the UK Engineering and Physical Sciences Research Council under grant EP/D067197/1.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 195–210, 2007.

systems may be deemed safe, but unfound erroneous states may arise lately in implementations. A common class of timelocks involve (action) deadlocks (e.g., due to mismatched blocking synchronisation). Usually, such timelocks may be found by checking deadlock-freedom; however, if a Zeno run is possible when the timelock occurs (e.g., due to some time-unconstrained loop), no deadlock will occur and the timelock will remain unnoticed.

Certain classes of timelocks can be avoided in a number of formal notations, including timed automata. For instance, in process algebras with asap [5], only internal actions can be made urgent, preventing timelocks that arise due to mismatched synchronisation. The same is achieved in Timed I/O Automata [6], Discrete Timed Automata [7], and Timed Automata with Deadlines [8,9], where construction ensures that either actions or delays are always possible. However, Zeno runs cannot be prevented by construction, as a suitable semantics would be too restrictive for the specifier. In Kronos [2] and Red [4], timelock-freedom can be verified as a liveness property. Kronos, Red and Profounder [10] verify properties only over time-divergent runs, i.e. algorithms must identify and discard Zeno runs in the process. Instead, more efficient algorithms may be used whenever absence of Zeno runs can be guaranteed in advance. Uppaal [3] does not distinguish time-divergent from Zeno runs during verification, but a liveness property can be verified to ensure that all runs are time-divergent. However, liveness checks can be very time-consuming (for instance, on-the-fly verification does not help, as the whole state space must be explored to confirm that Zeno runs cannot occur).

In [11,12], we refined Tripakis's strong non-Zenoness property [13] in a number of ways, which permitted an efficient check for absence of Zeno runs in simple timed automata models. Here we offer a more precise analysis of synchronisation and its effects on Zeno runs, and take into account Uppaal features such as non-zero clock updates, broadcast synchronisation and parametric templates. This results in a tool that is precise enough to assert absence of Zeno runs for a larger class of specifications than previously possible. When absence of Zeno runs cannot be inferred from the syntax of clock constraints and synchronisation, the analysis is inconclusive but returns all loops where Zeno runs may possibly occur. To benefit from this diagnosis, and avoid running liveness checks on the whole state space of the original network, we show how to obtain an abstract network that reduces the set of relevant behaviours to (an over-approximation of) those of unsafe loops. Then, by exploring the abstract network, liveness checks may assert absence of Zeno runs in the original network more efficiently. Experiments show that our analysis based on strong non-Zenoness, possibly followed by a liveness check on the abstract network, may be much more efficient than running liveness checks on the original network.

This paper is organised as follows. Section 2 introduces the timed automata model, and discusses timelocks and Zeno runs. Section 3 presents the static analysis based on strong non-Zenoness. Section 4 defines abstract networks. In Section 5 we present our tool and experimental results. We conclude the paper in Section 6, with suggestions for further research.

2 Timed Automata

Many extensions of Alur and Dill's model [1] have been proposed in the literature, and implemented in model-checkers. Uppaal's specification language, which we focus on in this paper, provides many features which facilitate the description of complex networks. Here we offer a brief account of the model (we refer to [3], and Uppaal's help files, for a more detailed presentation).

Uppaal automata are finite state machines (locations and edges), augmented with global (shared) clock and data variables, and synchronisation primitives. Concurrent systems are represented by networks of communicating automata. Concurrency is modeled by interleaving, and communication is achieved by synchronisation on channels and shared variables. Clocks range in the non-negative reals, and advance synchronously at the same rate (but may be updated independently). Edges denote instantaneous actions, and delays are possible only in locations. Clock and data variables can be used to constrain the execution of automata. Locations may be annotated with invariants, which constrain the allowed delays. Edges may be annotated with guards (enabling conditions), synchronisation labels (to distinguish observable from internal actions), and variable updates. Binary channels are blocking: matching input and output actions may only occur in pairs (`a?`/`a!`). More elaborate specifications can be obtained with the following features.

Variables. Available types include clocks, channels, bounded integers and Booleans, and arrays and record types can be defined over these types. Common arithmetic operators (and user-defined C-like functions) may be used in expressions. Clock constraints are (in)equalities between clocks (and clock differences) and integer expressions. Clocks can be assigned non-negative integer expressions.

Urgent and Committed locations. Urgent and committed locations disallow delays, forcing the immediate execution of enabled actions as soon as they are entered. In addition, committed locations restrict interleaving: only components that are currently in committed locations may execute enabled actions.

Urgent and Broadcast Channels. Synchronisation on urgent channels is binary, blocking, and must occur as soon as matching actions are enabled. Synchronisation on broadcast channels matches one output action with multiple input actions, and is non-blocking on the output side: input actions block until the output action is enabled, but the output action may be executed even if no input actions are enabled.

Templates and Selections. Parametric templates and selections provide a concise specification of similar components. A template provides an automaton and a number of parameters (bounded data variables), which can be read in the automaton's expressions (e.g., guards). Parameters are instantiated, generating multiple processes with the same control structure (the template's automaton). A selection denotes non-deterministic bindings between an identifier and values

in a given range. Selections annotate edges, which may use the identifiers in guards, synchronisation labels and updates. Every binding results in a different (instantiated) edge between the same two locations.

By way of example, Fig. 1 shows an Uppaal network to control access to a bridge for a number of trains (this model is a demo included with Uppaal's distributions; a similar model is explained in [3]). Incoming trains are represented by a `Train` template (left), access to the bridge is controlled by a `Gate` template (right). The `Train` template declares a local clock `x`, and an integer parameter `id` of type `id_t`$= [0..N-1]$, where N (a global constant) is the maximum number of incoming trains the gate controller may handle. For a simple example of the use of constraints and synchronisation, consider the edge `Cross` → `Safe`. The invariant `x<=5`, and guard `x>=3`, constrain the action to occur when `x` $\in [3, 5]$ (with the action being urgent when `x` $= 5$). Channel `leave` is binary, hence `Cross` → `Safe` and `Occ` → `Free` must occur simultaneously. To simulate and verify this network, Uppaal instantiates `id` to create N different `Train` processes, controlled by a single `Gate` process. This is modeled with arrays of channels (`leave`, `appr`, `stop` and `go`) and multi-transitions in `Gate`: the selection `e:id_t` makes `e` (a train identifier) range in $[0..N-1]$, providing a matching action for every `Train` process. The variables `e`, `len` (and an array `list`, not shown in the figure), and the functions `front()`, `enqueue()`, `dequeue()`, and `tail()`, manipulate a queue of approaching trains, some of which may be stopped (`stop!`) and later restarted (`go!`) to avoid collisions (e.g., the committed location, marked with `C`, ensures that the last approaching train is immediately stopped if a new train starts approaching the bridge).

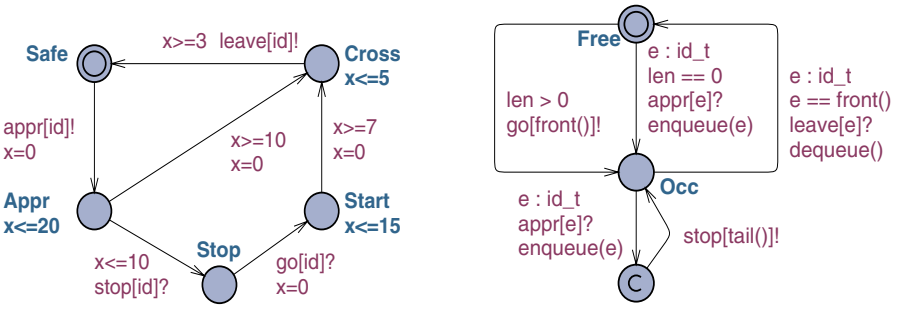


Fig. 1. An Uppaal network for a Train-Gate controller

2.1 Syntax and Semantics

We define a timed automata model which considers the main constructs of Uppaal's specification language. For the sake of simplicity, we omit an explicit formalisation of array and record types, functions, templates and selections.

Timed Automaton. Let $\mathcal{C}$ be a set of clocks, which range on the non-negative reals ($\mathbb{R}^{+0}$), and $\mathcal{D}$ be a set of data variables, which range in bounded domains

(e.g., Booleans and bounded integers). Let $\mathcal{V} = \mathcal{C} \cup \mathcal{D}$, and $Const(\mathcal{V})$ be the set of constraints (boolean expressions) on $\mathcal{V}$. Clock constraints are terms of the form $x \sim e$ or $x - y \sim e$, where $x, y \in \mathcal{C}$, $\sim \in \{<, >, =, \leq, \geq\}$ and e is an integer expression. We assume the usual set of arithmetic, Boolean and relational operators over data variables. Guards are conjunctions of clock constraints and boolean expressions not containing clocks. Invariants follow the syntax of guards but disallow lower bounds in clock constraints. Let $\mathcal{G}(\mathcal{V}) \subseteq Const(\mathcal{V})$ and $\mathcal{I}(\mathcal{V}) \subseteq Const(\mathcal{V})$ denote the sets of guards and invariants on $\mathcal{V}$, resp. Updates are (comma-separated) sequences of assignments of the form $var := e$, where $var \in \mathcal{V}$ and e is an expression. If var is a clock, then e must be a non-negative integer expression. Let $\mathcal{U}(\mathcal{V})$ denote the set of updates on $\mathcal{V}$. Synchronisation is defined over a set Ch of channels, some of which may denote urgent or broadcast channels. Let $UCh \subseteq Ch$ and $BCh \subseteq Ch$ denote all urgent and broadcast channels in Ch. Following restrictions in Uppaal, we require that edges labeled with urgent channels are not guarded with clock constraints. For any $a \in Ch$, observable actions may be labeled either with $a?$ (input, or receiving actions) or $a!$ (output, or emitting actions). Internal actions are labeled with ϵ. Let $Act = \{a?, a! \mid a \in Ch\} \cup \{\epsilon\}$ denote the set of all synchronisation labels.

A *timed automaton* is a tuple $A = (L, l_0, Lab, E, I, \mathcal{V})$. L is the set of locations, some of which may be tagged either as urgent or committed. Let $ULocs \subseteq L$, $CLocs \subseteq L$, and $NULocs = L \setminus (ULocs \cup CLocs)$ denote the subsets of urgent, committed and non-urgent locations in L ($ULocs \cap CLocs = \emptyset$). $l_0 \in L$ is the initial location, $Lab \subseteq Act$ is the set of synchronisation labels, $E \subseteq L \times Lab \times \mathcal{G}(\mathcal{V}) \times \mathcal{U}(\mathcal{V}) \times L$ is the set of edges, $I : NULocs \to \mathcal{I}(\mathcal{V})$ maps non-urgent locations to invariants, and $\mathcal{V}$ is a set of variables. Edges $(l, a, g, u, l') \in E$ are also denoted $l \xrightarrow{a,g,u} l'$, where a is the label, g is the guard and u is the update.

Semantics of a Network. A valuation maps clocks to non-negative reals, and data variables to corresponding domains. Let $\mathbb{V}(\mathcal{V})$ denote all possible valuations over $\mathcal{V}$, and let $\models$ denote constraint satisfiability over valuations. For any $v \in \mathbb{V}(\mathcal{V})$, and $\delta \in \mathbb{R}^+$, $v + \delta \in \mathbb{V}(\mathcal{V})$ is defined s.t. $\forall\, x \in \mathcal{C}.\, (v + \delta)(x) = v(x) + \delta$ and $\forall\, d \in \mathcal{D}.\ (v + \delta)(d) = v(d)$. Let $u \in \mathcal{U}(\mathcal{V})$ denote the sequence $var_1 := e_1, \ldots, var_m := e_m$, and $v_1 \in \mathbb{V}(\mathcal{V})$. We define $v_{i+1} \in \mathbb{V}(\mathcal{V})$, $1 \leq i \leq m$ s.t. $v_{i+1}(var_i) = [\![e]\!](v_i)$ and $v_{i+1}(var) = v_i(var)$ for any $var \in \mathcal{V} \setminus \{var_i\}$ ($[\![e]\!]_v$ denotes the value of expression e in v). Then, we define $u(v_1) = v_{m+1}$.

A *network* is a collection of automata $|A = |\langle A_1, \ldots, A_n\rangle$, where $A_i = (L_i, l_{i,0}, Lab_i, E_i, I_i, \mathcal{V}_i)$. The set of global (i.e., shared) variables of the network is given by $\bigcup_{1 \leq i \neq j \leq n} (\mathcal{V}_i \cap \mathcal{V}_j)$; all other variables are considered local to components. A location vector is denoted $\bar{l} = \langle l_1, \ldots, l_n\rangle$, where $l_i \in L_i$. We use $\bar{l}[l_i'/l_i]$ to denote that l_i in $\bar{l}$ is replaced by l_i'. For any set of indices J, we use $\bar{l}[(l_j'/l_j)_{j \in J}]$ to denote the replacement of l_j by l_j' in $\bar{l}$, for each $j \in J$. For $J = \{j_0, \ldots, j_m\}$ and $j_0 < j_1 < \ldots < j_m$, we use u_J to denote the sequential execution of updates $u_{j_0}, \ldots, u_{j_m}$. Let $\mathcal{V} = \bigcup_{i=1}^n \mathcal{V}_i$, $I = \bigwedge_{i=1}^n I_i$, and $CLocs = \bigcup_{i=1}^n CLocs_i$, where $CLocs_i \subseteq L_i$ is the set of committed locations of A_i. We use $CLocs(\bar{l})$ to denote the committed locations in $\bar{l}$.

The semantics of $|A$ are given by a timed transition system $(S, s_0, \{\epsilon\} \cup \mathbb{R}^+, T)$, where $S \subseteq L \times \mathbb{V}(\mathcal{V})$ is the set of reachable states (denoted $s = \langle \bar{l}, v \rangle$); $s_0 = \langle \bar{l}_0, v_0 \rangle$ is the initial state ($\bar{l}_0 = \langle l_{1,0}, \ldots, l_{n,0} \rangle$, $\forall\, var \in \mathcal{V}.\, v_0(var) = 0$); and $T \subseteq S \times \{\epsilon\} \cup \mathbb{R}^+ \times S$ is the transition relation. Action transitions are denoted $s \stackrel{\epsilon}{\Longrightarrow} s'$; delay transitions are denoted $s \stackrel{\delta}{\Longrightarrow} s'$ ($\delta \in \mathbb{R}^+$). We will use $s \stackrel{\epsilon}{\Longrightarrow}$ to denote any action transition from s. Transitions are computed:

1. *(from internal actions)* $\langle \bar{l}, v \rangle \stackrel{\epsilon}{\Longrightarrow} \langle \bar{l}[l'_i/l_i], u_i(v) \rangle$,
   for any $l_i \xrightarrow{\epsilon, g_i, u_i} l'_i \in T_i$ s.t. $v \models g_i$, $u_i(v) \models I(\bar{l}[l'_i/l_i])$, and $l_i \in CLocs_i$ or $CLocs(\bar{l}) = \emptyset$.
2. *(from actions emitting on broadcast channels)* $\langle \bar{l}, v \rangle \stackrel{\epsilon}{\Longrightarrow} \langle \bar{l}[l'_i/l_i], u_i(v) \rangle$,
   for any $l_i \xrightarrow{b!, g_i, u_i} l'_i \in T_i$ s.t. $b \in BCh$, $v \models g_i$, $u_i(v) \models I(\bar{l}[l'_i/l_i])$, there is no $l_j \xrightarrow{b?, g_j, u_j} l'_j \in T_j$ $(j \neq i)$ s.t. $v \models g_j$, and $l_i \in CLocs_i$ or $CLocs(\bar{l}) = \emptyset$.
3. *(from binary synchronisation)* $\langle \bar{l}, v \rangle \stackrel{\epsilon}{\Longrightarrow} \langle \bar{l}[l'_i/l_i, l'_j/l_j], u_j(u_i(v)) \rangle$,
   for any $l_i \xrightarrow{a!, g_i, u_i} l'_i \in T_i$ and $l_j \xrightarrow{a?, g_j, u_j} l'_j \in T_j$ $(j \neq i)$ s.t. $a \notin BCh$, $v \models g_i \wedge g_j$, $u_j(u_i(v)) \models I(\bar{l}[l'_i/l_i, l'_j/l_j])$, and $\{l_i, l_j\} \cap CLocs(\bar{l}) \neq \emptyset$ or $CLocs(\bar{l}) = \emptyset$.
4. *(from broadcast synchronisation)* $\langle \bar{l}, v \rangle \stackrel{\epsilon}{\Longrightarrow} \langle \bar{l}[l'_i/l_i, (l'_j/l_j)_{j \in J}], u_J(u_i(v)) \rangle$,
   for any $l_i \xrightarrow{b!, g_i, u_i} l'_i \in T_i$ s.t. $b \in BCh$, $J \subseteq [1..n] \setminus \{i\}$ is the maximal set of indices s.t. for any $j \in J$ there is a $l_j \xrightarrow{b?, g_j, u_j} l'_j \in T_j$, where $v \models g_i \wedge \bigwedge_j g_j$, $u_J(u_i(v)) \models I(\bar{l}[l'_i/l_i, (l'_j/l_j)_{j \in J}])$, and $(\{l_i\} \cup \{l_j \mid j \in J\}) \cap CLocs(\bar{l}) \neq \emptyset$ or $CLocs(\bar{l}) = \emptyset$.
5. *(from delays)* $\langle \bar{l}, v \rangle \stackrel{\delta}{\Longrightarrow} \langle \bar{l}, v + \delta \rangle$,
   for any $\delta \in \mathbb{R}^+$ s.t. $(v + \delta) \models I(\bar{l})$, $CLocs(\bar{l}) \cup ULocs(\bar{l}) = \emptyset$, and no transition $\langle \bar{l}, v + \delta' \rangle \stackrel{\epsilon}{\Longrightarrow}$ $(\delta' < \delta)$ can be computed from synchronisation over urgent channels (either by rules 2,3 or 4).

Runs, Zeno Runs and Timelocks. A *run* is a path in the timed transition system, $\rho \triangleq s_1 \stackrel{\gamma_1}{\Longrightarrow} s_2 \stackrel{\gamma_2}{\Longrightarrow} \ldots$, where $s_i \in S$, $\gamma_i \in \{\epsilon\} \cup \mathbb{R}^+$, s.t. ρ ends in some state $s_n \in S$ (if ρ is finite). A run ρ is *time-divergent* if the sum of all delays occurring in ρ is infinite. A *Zeno run* performs infinitely many actions in finite time. A *timelock* is a state where time-divergent runs are not possible. Absence of Zeno runs does not guarantee timelock-freedom, or vice versa. However, both absence of Zeno runs and deadlocks suffice to guarantee timelock-freedom (detailed presentations of timelocks and Zeno runs can be found in [9,11,12]).

Figure 2 illustrates the occurrence of timelocks and Zeno runs. Assume that all loops belong to different components, and that all clock values are initially zero. Consider the network `Net1`. If `L1` → `L2` does not occur early enough, the loop at `L3` will not have the chance to synchronise. Eventually, the invariant `y<=1` will prevent further delays and the entire network will block: a *time-actionlock* has occurred. A *Zeno-timelock*, where the only possible infinite runs are Zeno runs, occurs if the loop in `L3` synchronises with `L2` → `L1`: Eventually, the network will reach a state where the invariant `z<=3` prevents further delays, but a Zeno run is induced by synchronisation between the loops in `L3` and `L4` (`z` is never

reset). Note that, in any case, the loop in `T` cannot iterate more than three times. In contrast, in `Net2`, the loop in `L1` exhibits Zeno runs but delays are always possible in other runs; the loop in `T` may always iterate (once per time unit). However, Zeno runs make `Net2` unfair to `T`: if the loop in `L1` would not be able to iterate arbitrarily fast, fairness would be guaranteed by the invariant `t<=1`.

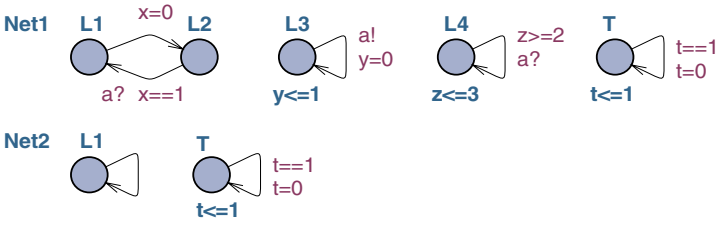


Fig. 2. Timelocks, Zeno runs, and test automata

Verifying Time-Divergence in Uppaal. In Uppaal, both absence of timelocks and Zeno runs can be characterised by a liveness formula, defined over an extended network augmented with a test automaton [14]. Figure 2 illustrates the approach, where the loop in `T` represents the test automaton. Uppaal leads-to operator [3], `-->`, can be used to define the formula $\lambda_U \triangleq$ `t==0 --> t==1`, which is satisfiable when any (`t==0`)-state is eventually followed by a (`t==1`)-state in every run. Equivalently, λ_U is satisfiable if *all* runs in the original network are time-divergent. For instance, λ_U is not satisfiable in `Net1` in Fig. 2 (and, perhaps against our intuition, neither is it satisfiable in `Net2`).

Test automata, and corresponding λ_U properties, give us a simple, robust way to verify time-divergence. However, this approach has a number of disadvantages. Liveness verification is computationally expensive in general (it requires a form of nested reachability analysis), and in the case of λ_U, it is likely to suffer from state-explosion (absence of Zeno runs cannot be confirmed unless the whole state space has been explored, i.e., on-the-fly verification will not give any benefits). Unfortunately, Uppaal also disallows symmetry reduction [15] for leads-to formulas. A further limitation of λ_U is that, in networks with timelocks, it will fail to hold even if Zeno runs do not occur. This may be problematic whenever timelocks are intentionally introduced. For instance, "sink" committed locations were used to enforce global termination in the protocol of [16], which is nonetheless free from Zeno runs (see our notes on the `lipsync` case study, in Sect. 5).

3 Strong Non-zenoness

Strong non-Zenoness [13] is a static property of loops (defined as cycles in the automaton's graph), which suffices to guarantee absence of Zeno runs. A loop is strongly non-Zeno (SNZ) if, from the syntax of guards and updates, a clock can be inferred to be bounded from below ($x \geq n, n \geq 1$) and reset ($x := 0$)

in the loop. A network is free from Zeno runs if all loops are SNZ [13] (strong non-Zenoness guarantees cumulative n-delays between loop iterations).

In [11,12], we obtained weaker conditions to guarantee absence of Zeno runs in a network: (a) it suffices to consider loops that correspond to elementary cycles (i.e., cycles with exactly one repeating location), and (b) Zeno runs cannot occur if all NSNZ loops have at least one observable action, which cannot be matched against any other NSNZ loop (blocking synchronisation with any SNZ loop guarantees time-divergence). Our analysis in [11,12] was able to assert absence of Zeno runs for a larger class of specifications (w.r.t. [13]), but it assumes a simple timed automata model. In what follows, we show that the analysis is not sound when Uppaal extensions such as non-zero clock assignments and broadcast channels are considered, and that synchronisation can be better exploited to improve precision. On the other hand, the analysis is insensitive to urgent and committed locations and urgent channels (urgent actions cannot make a SNZ loop iterate faster than its witness clock permits), and it presents advantages when parameters and selections are ignored. These issues are taking into account to define a more comprehensive analysis on Uppaal networks (Sect. 3.1).

Non-zero Clock Assignments. If a clock x is assigned a non-zero value in a loop, then Zeno runs may occur even if x is a witness for strong non-Zenoness. For instance, in Fig. 3, a Zeno run may occur in `lp1` if `x=4` occurs immediately after `x=0`. On the other hand, `x` is a SNZ witness for `lp2`: `x=4` has no effect on `x>3` because `x=0` occurs after it. Similarly, `x` is a witness for `lp3`: time must necessarily pass to enable `x>3` after `x=1` has occurred. In general, we must ensure that no conflicting updates may occur between an update and a lower bound, also taking into account that the SNZ witness may be a shared variable.

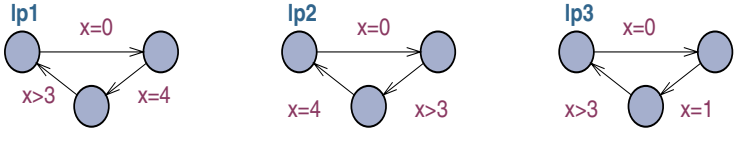


Fig. 3. Non-zero clock updates

Broadcast Channels. Due to non-blocking semantics, a NSNZ loop that emits on a broadcast channel may complete its iterations even if the receivers are not enabled. Thus, in general, synchronisation between NSNZ and SNZ loops may not be free from Zeno runs. For instance, in Fig. 4 (where `b` is a broadcast channel and `a` is binary channel), Zeno runs may occur between `lp1` and `lp2`. Zeno runs cannot occur between `lp3` and `lp4` (input actions are always blocking), or between `lp5`, `lp6` and `lp7` (a SNZ loop is matched on a binary channel).

Synchronisation of Multiple Loops. Whenever two NSNZ loops have at least a pair of matching observable actions, the analysis in [12] is unable to guarantee absence of Zeno runs. However, the occurrence of Zeno runs may nonetheless be prevented if the NSNZ loops need a SNZ loop to complete their iterations.

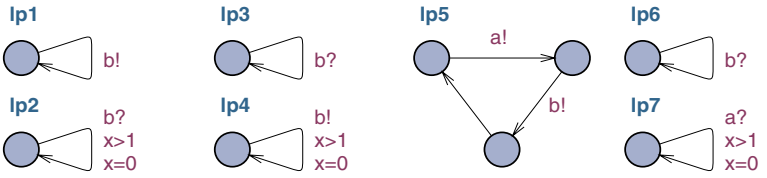


Fig. 4. Broadcast channels

This suggests that a more precise analysis can be obtained, which finds groups of NSNZ loops that may not need to match with SNZ loops to complete their iterations (and thus, to exhibit Zeno runs).

Templates and Selections. In many cases, the analysis of strong non-Zenoness will benefit from parametric templates and selections: SNZ witnesses may be computed directly from the template's structure, regardless of the actual parameter values and selection bindings. Consider again the network of Fig. 1. All loops in `Train` are SNZ due to guards and updates on `x`, i.e., parameters and selection bindings can be ignored. Loops in `Gate` are NSNZ, but they may synchronise only with loops in `Train`, on binary channels. Thus, the network is free from Zeno runs. Furthermore, strong non-Zenoness was guaranteed regardless of actual parameter values, which allows us to infer that the network is safe for any number of trains. In contrast, λ_U may only assert absence of Zeno runs for a fixed number of trains (with limitations in scalability).

3.1 Strong Non-zenoness for Uppaal Networks

Let A be a timed automaton (Sect. 2.1). A *loop* is an elementary cycle in A; i.e., a sequence $\langle l_0 \xrightarrow{a_1,g_1,r_1} l_1 \cdots l_{n-1} \xrightarrow{a_n,g_n,r_n} l_n \rangle$, where $l_0 = l_n$ and $l_i \neq l_j$ for all $0 \leq i \neq j < n$. An *observable loop* is one that contains observable actions (otherwise, it is an *internal loop*). We say that a run *covers* a loop when it visits all its edges infinitely often. For any clock constraint ϕ and guard g, $\phi \in g$ denotes that ϕ can be inferred from g. For any clock x, $m \in \mathbb{N}$, and update u, $x := m \in u$ denotes that the value of x is m after all assignments in u are sequentially executed. Let e be an edge in lp with guard g, a clock x occurring in g, and $n \in \mathbb{N}, n > 0$. We use $x_{\sqsupseteq n} \in g$ to denote either $x \sqsupseteq n \in g$ ($\sqsupseteq \in \{=, >, \geq\}$), or $x - y \sqsupseteq n \in g$ ($\sqsupseteq \in \{>, \geq\}$); we use $x_{\geq n}$ if $x - y = n \in g$. We say that $x_{\sqsupseteq n}$ ($\sqsupseteq \in \{=, >, \geq\}$) is the lower bound for x in g if $x_{\sqsupseteq n} \in g$ and there is no $n' > n$ s.t. $x_{\sqsupseteq n'} \in g$. A refined strong non-Zenoness property can be defined as follows.

Definition 1. (SNZ loop) *A loop lp is strongly non-Zeno (SNZ) if there is a clock x and two edges e_1, e_2 in lp, where u is the update in e_1 and g is the guard in e_2, $x := m \in u$, $x_{\sqsupseteq n}$ is the lower bound for x in g, $m < n$, and there is no $x := m'$, $m' \geq n$, in any update in the loop in the path from e_1 to e_2. We refer to x as a (SNZ) witness of lp.*

Definition 2. (Safe loop) *A loop lp is safe if (a) lp is a SNZ loop, and (b) lp has a local witness, or every loop that may update any witness of lp is a SNZ loop with a local witness.*

Proposition 1. *If lp is a safe loop, any run that covers lp is time-divergent.*

Proof. By definition, a SNZ witness clock x guarantees time-divergence for any run that covers lp, unless x is externally updated infinitely often, and with delay $\delta < 1$ between updates. Such conflicting updates cannot happen if lp is safe. □

Definition 3. (Synchronisation Group) *Let UL_{sync} be the set of all unsafe observable loops in a network $|A$. A synchronisation group (or sync group, for short) is a maximal, non-empty set $S \subseteq UL_{sync}$, s.t. for any $lp \in S$, and any observable action in lp that synchronises on a binary channel or emits on a broadcast channel, there is a matching action in some $lp' \in S$.*

For example, for the network shown in Fig. 5, the analysis returns only the group {`lp3`, `lp4`}: the loops `lp2` and `lp3` cannot synchronise together (not without `lp1`, which is a safe loop and therefore not considered for grouping).

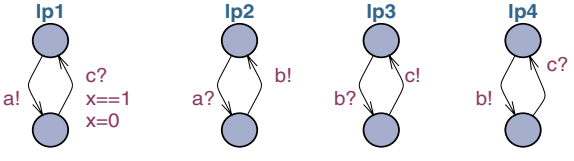


Fig. 5. Synchronisation Groups

Proposition 2. *If a Zeno run occurs which covers an observable loop lp, then there is at least one sync group S s.t. $lp \in S$.*

Proof. Any Zeno run contains a suffix ρ that only visits edges of a set UL of unsafe loops, infinitely often [12]. Let $lp \in UL$ be an observable loop. There is some $UL' \subseteq UL$, $lp \in UL'$, composed entirely by observable loops that synchronise together (otherwise, ρ could not cover lp). Necessarily, for any observable action with synchronisation label lb in any loop in UL', there is a loop in UL' with a matching action (if $lb = a?$ or $lb = a!$, where a is a binary channel, or $lb = b?$, where b is a broadcast channel). By definition, every loop in UL' must be part of a sync group. □

Note that, synchronisations that are subject to Zeno runs will be identified by sync groups (i.e., the analysis is conservative), but a sync group may represent synchronisations that are not possible at run-time (e.g., due to the ordering of observable actions in a loop or unreachable valuations).

Proposition 3. *If all internal loops in a network are safe, and no sync groups can be formed, the network is free from Zeno runs.*

Proof. Follows from Props. 1 and 2. □

Implementation Notes. Templates are largely regarded as component automata, in the sense of Sec. 2.1. Lower bounds and clock assignments are inferred from the syntax of guards and updates, whenever constant values can be directly computed (in general, witnesses may not be extracted where variables, parameters, functions or selection identifiers occur). For any array of channels, `a` say, we assume that `a[`e_i`]!` and `a[`e_j`]?` match, unless e_i and e_j can be resolved to different constant values. Sync groups may include unsafe loops in parametric templates, whose observable actions match other actions in the loop or other loops in the template (albeit rare in practice, this is permitted in Uppaal).

4 Helping Liveness Checks

In some networks, data constraints prevent Zeno runs from occurring, even though unsafe loops may be found. Moreover, any Zeno run has a suffix that visits edges of unsafe loops only, and does so infinitely often [12]. Therefore, liveness checks could in principle assert absence of Zeno runs by exploring solely the behaviour of unsafe loops. In particular, our aim is to reduce the state space that Uppaal needs to explore to verify the λ_U property (Sect. 2). Given a network $|A$, and a set of unsafe loops UL found by static analysis (Sect. 3), an abstract network $\alpha(|A, UL)$ can be obtained that contains just the loops in UL. In addition, $\alpha(|A, UL)$ provides the necessary valuations that allow the loops in UL to behave as they do in $|A$, such that, if a liveness check ensures the absence of Zeno runs in $\alpha(|A, UL)$, then this also holds for $|A$. Our aim is to offer an static abstraction, hence, the valuations that can be reached in $|A$ at the loops' entry locations may have to be over-approximated. A consequence of this over-approximation is that liveness checks will be sufficient-only: Zeno runs may be found in $\alpha(|A, UL)$ which do not occur in $|A$. Hence, the precision of the abstraction depends on how accurately we can approximate the relevant valuations at entry locations.

For example, Fig. 6(left) shows a template for processes running the Fischer's mutex protocol. Declarations are as follows: `x` is a local clock, `pid` and `k` are integer parameters (`pid`, `k` > 0), and `id` is a global integer variable. The only unsafe loop (a NSNZ loop) is $\langle$`req` $\rightarrow$ `wait` $\rightarrow$ `req`$\rangle$, and it is free from Zeno runs. This loop may complete iterations only after `id = 0`, but this may not occur arbitrarily fast (the update occurs only in the SNZ loops of competing processes). The abstract network (Fig. 6, right) needs to consider only the NSNZ loop, and provides initial valuations so the loop may behave as in the original network. The liveness check may now work on a reduced state space, where the loop cannot complete a single iteration (thus, the abstract network is free from Zeno runs, and so is the original network). Section 5 shows that the abstract network allows a more efficient check (see entries for `fischer` and `fischerABS` in Table 1).

4.1 Abstract Network

Let $|A = \langle A_1, \ldots, A_n\rangle$ be a network, s.t. $UL \neq \emptyset$ is the set of unsafe loops in $|A$. Let $UL_1, \ldots, UL_m$ be a partition of UL w.r.t. network components, i.e.

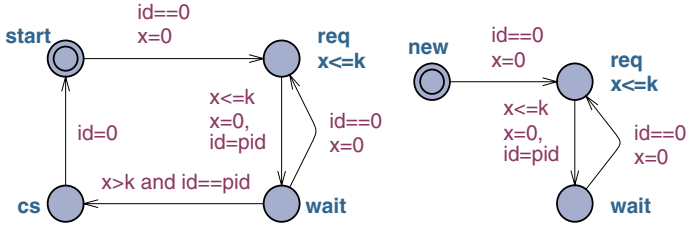


Fig. 6. A process template in Fischer's protocol: original (left) and abstract (right)

$UL = \bigcup_{j=1}^{m} UL_j$ and all loops in UL_j $(1 \leq j \leq m)$ belong to the same component A_i $(1 \leq i \leq n)$ (we define $|A(UL_j) = A_i$). For any loop $lp \in UL$, $L(lp)$ is the set of locations, $E(lp)$ is the set of edges, $Lab(lp)$ is the set of labels, and $\mathcal{V}(lp)$ is the set of variables occurring in lp. Let $Entry(lp)$ be the set of locations of lp that are either a component's initial location, or have at least one ingoing edge that is not part of any $lp' \in UL$. From every UL_j, we will define a component of the abstract network that includes all loops in UL_j, and provides the necessary valuations so that loops in UL_j may be entered and behave as they do in $|A$.

A variable is *read* in lp if it occurs in a guard, invariant or rhs-expression of an assignment in lp. A variables is *set* in lp if it occurs in the lhs-expression of an assignment in lp. A variable is *used* in lp if it occurs in a guard or invariant in lp, or it occurs in the rhs-expression of an assignment in lp, whose lhs-expression involves a variable that is used in $lp' \in UL$. Let $R(lp, l) \subseteq \mathcal{V}(lp)$ be the set of variables that are used in lp, and are read before they are set (or just read) in any edge of lp, in the path starting at $l \in Entry(lp)$. $R(lp, l)$ denotes a group of variables which may cause Zeno runs, if lp is entered at l; for such variables, the valuations that are reachable in $|A$ should also be reachable in the abstract network. On the other hand, whenever lp is entered at l, the values of variables that are not in $R(lp, l)$ are irrelevant to the occurrence of Zeno runs, and the abstract network may simply provide default initial values.

For each $lp \in UL_j$, let $Clocks(lp)$ be the set of clocks that occur in lp, and $Clocks(UL_j) = \bigcup_{lp \in UL_j} Clocks(lp)$, where $|Clocks(UL_j)| = k$. Let $NewL(UL_j) = \{loc_0, \ldots, loc_k\}$ be a set of new (non-urgent) locations. We define a set of edges,

$$ResE(UL_j) = \{(loc_i, \epsilon, true, x := 0, loc_{i+1}) \,|\, 0 \leq i < k, x \in Clocks(UL_j)\}$$

which allows loops in UL to be entered with valuations where two or more clocks differ (otherwise, all clocks will have the same value when loops are entered, possibly missing valuations that are reachable in $|A$ and are the cause for Zeno runs). We also assume that a set of edges $InitE(lp)$ can be defined, which connect loc_k with every $l \in Entry(lp)$. For every variable in $R(lp, l)$, edges in $EntryE(lp)$ must provide an over-approximation of the valuations that are reachable at l in $|A$. We do not prescribe here the guards and updates that will define such edges: approximations should be informed by the syntax of the network at hand, both to obtain a static abstraction and to provide accurate valuations (as much as possible). For instance, in Uppaal, selections may be used to assign a range

of values to data variables (ranges may represent, in the worst case, the entire domain), and extrapolation will implicitly generate reachable clock valuations (Uppaal compute the possible delays at each location vector, during verification).

From UL_j with $|A(UL_j) = (L, l_0, Lab, E, I, \mathcal{V})$, we derive a component of the abstract network, $\alpha(UL_j) = (L', loc_0, Lab', E', I', \mathcal{V}')$, where

- $L' = NewL(UL_j) \cup \bigcup_{lp \in UL_j} L(lp)$
- $Lab' = \{\epsilon\} \cup \bigcup_{lp \in UL_j} Lab(lp)$
- $E' = ResE(UL_j) \cup \bigcup_{lp \in UL_j} (E(lp) \cup EntryE(lp))$
- $I' = I/L' \cup \{(l, true) \,|\, l \in NewL(UL_j)\}$ (I/L' denotes I restricted to L')
- $\mathcal{V}' = \bigcup_{lp \in UL_j} \mathcal{V}(lp)$

Finally, the abstract network is given by $\alpha(|A, UL) = \langle \alpha(UL_1), \ldots, \alpha(UL_m) \rangle$.

Proposition 4. *$|A$ is free from Zeno runs if $\alpha(|A, UL)$ is free from Zeno runs.*

Proof. For any $lp \in UL$, $l \in Entry(lp)$, and $var \in R(lp, l)$, every valuation that is reachable in $|A$ at l is also reachable in $\alpha(|A, UL)$. These valuations are over-approximated either by edges in $ResE(UL_j)$ or $EntryE(lp)$, or generated by clock extrapolation. On the other hand, if lp is entered at l, then the values of any $var \in \mathcal{V} \setminus R(lp, l)$ may not be related in $\alpha(|A, UL)$ and $|A$; however, the iterations of any $lp \in UL$ will not depend on such initial values. Consider now a Zeno run ρ in $|A$ s.t. (a) ρ only visits edges of a set of loops $UL' \subseteq UL$, and does so infinitely often; and (b) any variable that does not occur in UL' has a constant value along ρ (we can prove that if a Zeno run occurs in $|A$, then ρ is a suffix of that run [12]). Let ρ' be the Zeno run that is the projection of ρ onto UL. Let $s = \langle \bar{l}, v \rangle$ be any state of ρ'. Necessarily, for any variable var in UL, either (a) $v(var)$ is the value at some entry location l of some $lp \in UL$, or $v(var)$ can be derived from a valuation reachable at l by visiting a sequence of edges in UL', or (b) the value of var is irrelevant to iterations of loops in UL'. By construction of the abstract network, we can infer the occurrence of a Zeno run in $\alpha(|A, UL)$, which is similar to ρ' except possibly for the value of any var that never prevents iterations or enforces $\delta \geq 1$ delays. □

Implementation Notes. In certain cases, it may be necessary to convert urgent or committed locations to non-urgent locations, to avoid spurious timelocks in the abstract network. This over-approximates the valuations reachable at the location in question and, therefore, does not compromise the conservative nature of our abstraction.

5 Experimental Results

Our ZenoChecker tool (ZC) implements the analysis of Sect. 3.1 over Uppaal networks. ZC runs a cycle detection algorithm [17], checks strong non-Zenoness, and identifies sync groups. All unsafe internal loops, and observable loops in any

sync group, are grouped by template and returned as an Uppaal network. The network is free from Zeno runs if no such loops are found.

Table 1 compares the analysis performed by ZC, with verification of λ_U in Uppaal (Sect. 2). Very efficient tests were obtained either from ZC alone, or from λ_U verified on abstract networks (when ZC found unsafe loops). For `gbox`, `train-gate-q` and `fischer`, the returned unsafe loops readily revealed that Zeno runs could not occur (we have included the abstraction just for comparison purposes). Uppaal found a timelock in `lipsync`, hence it could not inform on the occurrence of Zeno runs. It turns out that the timelock is intentional in `lipsync` [16], and ZC was able to confirm that the network is free from Zeno runs. In the abstract networks obtained for `train-gate-q`, `fischer`, `yahalom` [18] and `zeroconf` [19], committed and urgent locations were made non-urgent, and a number of unsafe loops were removed before λ_U was verified (it was evident that the removed loops could not contribute to the occurrence of Zeno runs). A timelock was found in `yahalom`, and `yahalomABS` was used to refine the analysis and show that Zeno runs could also occur. Zeno runs were found both `yahalomABS` and `zeroconfABS`; as the abstractions are conservative, this

Table 1. Checking absence of Zeno runs. Performance figures are rounded up, and were obtained with `memtime` (`www.uppaal.com`), running on 2 Pentium 3, 1.4GHz processors, 1GB RAM, Debian Linux 2.6.8. Uppaal's `verifyta` executed with default options. ZC: our tool; λ_U: liveness check in Uppaal; $\perp$: aborted (out of time); - = not applicable; ?: inconclusive (N = number of NSNZ loops found by ZC); ✓: free from Zeno runs; P: parameter value; *ABS: abstract network. For `fddi` and `csmacd-u`, all instances were modeled by different, non-parametric templates (e.g., `csmacd-u32` denotes the csma/cd protocol with 32 competing stations).

Network	Time (sec)		RSS (MB)		VSize (MB)		Result	
	ZC	λ_U	ZC	λ_U	ZC	λ_U	ZC	λ_U
gbox	1	11644	16	25	256	68	? (7)	sat
gboxABS	-	1	-	1	-	2	-	sat
lipsync	1	1	16	3	256	53	✓	not sat(?)
train-gate(P=8)	1	1496	16	715	256	762	✓	sat
bocdp	890	$\perp$	21	$\perp$	260	$\perp$	✓	sat
fddi(32/4)	2	$\perp$	18	$\perp$	255	$\perp$	✓	sat
csmacd	1	5204	15	16	255	60	✓	sat
watersystem	1	1897	16	41	256	82	✓	sat
train-gate-q(P=8)	1	2361	15	949	256	1335	? (1)	sat
train-gate-qABS(P=8)	-	1	-	1	-	1	-	sat
fischer(P=6)	1	$\perp$	14	$\perp$	256	$\perp$	? (1)	sat
fischerABS(P=6)	-	3665	-	17	-	63	-	sat
csmacd-u32	1	1	18	54	256	5	? (97)	not sat
bmp	1	1	15	1	256	1	? (4)	not sat
yahalom	1	1	15	1	255	2	? (13)	not sat(?)
yahalomABS	-	1	-	1	-	2	-	not sat
zeroconf	1	1676	16	274	256	315	? (15)	not sat
zeroconfABS	-	1	-	1	-	1	-	not sat(?)

required close examination of the networks to ensure that the Zeno runs were not spurious.

6 Conclusions

We discussed an efficient analysis of Zeno runs on timed automata (based on Tripakis' strong non-Zenoness property), tailored to Uppaal's rich specification language. This is implemented in a tool, which accepts Uppaal networks and finds all possible loops where Zeno runs may occur. The analysis is static and thus conservative; if no unsafe loops are found the network is free from Zeno runs, but this may be the case even if unsafe loops are found. A good tradeoff between precision and efficiency is achieved thanks to a refined definition of strong non-Zenoness, and a comprehensive examination of synchronisation scenarios. In general, whenever the static analysis is inconclusive, absence of Zeno runs must be verified through liveness properties. This verification, however, can hardly avoid state-explosion. We improved this case with an abstraction that reduces the original network to (an approximation of) the behaviours of unsafe loops. In this way, a liveness check may be spared much of the state space where Zeno runs cannot occur. Positive experimental evidence was obtained, which showed that the combined approach of static analysis and abstraction may be much more efficient than direct liveness verification, and yet precise enough to assert absence of Zeno runs. As future work, our tool could be extended to infer absence of Zeno runs from expressions where parameters and data variables occur, given that data domains are bounded in Uppaal. In addition, as the analysis is insensitive to urgent behaviour, it could be adapted easily to deal with Timed Automata with Deadlines [8,9] (where absence of Zeno runs imply timelock-freedom).

Acknowledgments. We are grateful to the researchers who made the benchmark models available, and to the reviewers for their insightful comments.

References

1. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
2. Yovine, S.: Kronos: A verification tool for real-time systems. International Journal of Software Tools for Technology Transfer 1(1-2), 123–133 (1997)
3. Berhmann, G., David, A., Larsen, K.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
4. Wang, F.: Model-checking distributed real-time systems with states, events, and multiple fairness assumptions. In: Rattray, C., Maharaj, S., Shankland, C. (eds.) AMAST 2004. LNCS, vol. 3116, pp. 553–568. Springer, Heidelberg (2004)
5. Regan, T.: Multimedia in temporal LOTOS: A lip synchronisation algorithm. In: PSTV XIII, 13th Protocol Spec. Testing & Verification, North-Holland, Amsterdam (1993)

6. Gebremichael, B., Vaandrager, F.: Specifying Urgency in Timed I/O Automata. In: Proceedings of SEFM 2005, pp. 64–73. IEEE Computer Society Press, Los Alamitos (2005)
7. Gomez, R., Bowman, H.: Discrete timed automata and MONA: Description, specification and verification of a multimedia stream. In: König, H., Heiner, M., Wolisz, A. (eds.) FORTE 2003. LNCS, vol. 2767, pp. 177–192. Springer, Heidelberg (2003)
8. Bornot, S., Sifakis, J., Tripakis, S.: Modeling urgency in timed systems. In: de Roever, W.-P., Langmaack, H., Pnueli, A. (eds.) COMPOS 1997. LNCS, vol. 1536, pp. 103–129. Springer, Heidelberg (1998)
9. Bowman, H.: Time and action lock freedom properties for timed automata. In: Proceedings of FORTE 2001, pp. 119–134. Kluwer Academic Publishers, Dordrecht (2001)
10. Tripakis, S., Yovine, S., Bouajjani, A.: Checking Timed Büchi Automata emptiness efficiently. Formal Methods in System Design 26(3), 267–292 (2005)
11. Gomez, R.: Verification of Real-Time Systems: Improving Tool Support. PhD thesis, Computing Laboratory, University of Kent (October 2006)
12. Bowman, H., Gomez, R.: How to stop time stopping. Formal Aspects of Computing 18(4), 459–493 (2006)
13. Tripakis, S.: Verifying progress in timed systems. In: Katoen, J.-P. (ed.) AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999. LNCS, vol. 1601, Springer, Heidelberg (1999)
14. Aceto, L., Bouyer, P., Burgueño, A., Larsen, K.: The power of reachability testing for timed automata. Theoretical Computer Science 1-3(300), 411–475 (2003)
15. Hendriks, M., Behrmann, G., Larsen, K., Niebert, P., Vaandrager, F.: Adding symmetry reduction to UPPAAL. In: Larsen, K., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 46–59. Springer, Heidelberg (2004)
16. Bowman, H., Faconti, G., Katoen, J.P., Latella, D., Massink, M.: Automatic verification of a lip synchronisation protocol using UPPAAL. Formal Aspects of Computing 10(5-6), 550–575 (1998)
17. Szwarcfiter, J., Lauer, P.: A search strategy for the elementary cycles of a directed graph. BIT 16, 192–204 (1976)
18. Corin, R., Etalle, S., Hartel, P.H., Mader, A.: Timed model checking of security protocols. In: Proceedings of FMSE '04, pp. 23–32. ACM Press, New York (2004)
19. Gebremichael, B., Vaandrager, F., Zhang, M.: Analysis of the zeroconf protocol using Uppaal. In: Proceedings of EMSOFT '06, pp. 242–251. ACM Press, New York (2006)

Partial Order Reduction for Verification of Real-Time Components

John Håkansson[1] and Paul Pettersson[1,2]

[1] Dept. of Information Technology, Uppsala University, P.O. Box 337,
SE-751 05 Uppsala, Sweden
{johnh,paupet}@it.uu.se
[2] Dept. of Computer Science and Electronics, Mälardalen University, P.O. Box 883,
SE-721 23, Västerås, Sweden
Paul.Pettersson@mdh.se

Abstract. We describe a partial order reduction technique for a real-time component model. Components are described as timed automata with data ports, which can be composed in static structures of unidirectional control and data flow. Compositions can be encapsulated as components and used in other compositions to form hierarchical models. The proposed partial order reduction technique uses a local time semantics for timed automata, in which time may progress independently in parallel automata which are resynchronized when needed. To increase the number of independent transitions and to reduce the problem of re-synchronizing parallel automata we propose, and show how, to use information derived from the composition structure of an analyzed model. Based on these ideas, we present a reachability analysis algorithm that uses an ample set construction to select which symbolic transitions to explore. The algorithm has been implemented as a prototype extension of the real-time model-checker Uppaal. We report from experiments with the tool that indicate that the technique can achieve substantial reduction in the time and memory needed to analyze a real-time system described in the studied component model.

1 Introduction

Component-based development has been successfully used for desktop and e-business applications, and it is currently being introduced in many embedded systems. The resource constrained nature of these systems has motivated the development of specific component models [1,14,16,21] and formal frameworks, e.g. [8,9,11].

In general, a component based system is a composition of components, where a component is an open system that accepts input from its environment and produces output. The internal behaviour of a component can be described by a composition, thereby forming a hierarchy of compositions. Components interact with their environment through ports, according to interfaces defined for the ports. Figure 1 shows three components A, B and C. The two components A

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 211–226, 2007.

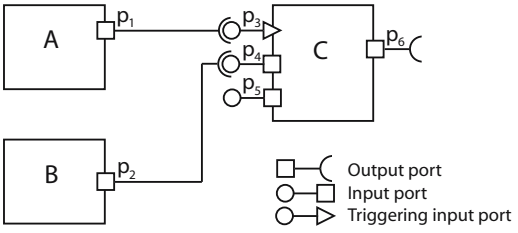


Fig. 1. An example of composition where components A, B and C are composed by connecting port p_1 to p_3, and p_2 to p_4

and B each have an output port, while component C has three input ports and an output port. Components can be composed into more complex functional units with well defined interfaces. A *horizontal composition* is a set of components with their ports connected, as in Fig. 1. The connections define how components can interact within a composition. A *vertical composition* is a component with its internal behaviour defined by a horizontal composition.

We use a model for components and composition based on the SAVE[1] component model [1,5], and designed for vehicle applications with analysability and safety in mind. The component model is similar to IEC 1131 [16] and Rubus [16]. In our model a component is either idle or executing, and data is transferred from a component when its execution has finished. Some input ports are called trigger ports, and are used to trigger the transition of a component's state from idle to executing. Control flow is specified by means of trigger ports: when one component becomes idle, it can trigger other components so that they become executing. As timeliness is an important property for many embedded systems, we model the execution of components as timed automata [2].

Model checking is an well-established and popular approach for analysis of models, although it is inherently complex and suffers from the so-called state-space explosion problem [13]. Partial order reduction [10,20,18,7] has been suggested as a technique to reduce the state-space explosion caused by parallelism. The idea is to explore representative traces — a property preserving subset of the full model based on independence of transitions. In this paper we present a partial order reduction technique for real-time systems, which is guided by the structure the component based system being analyzed. As in [3,17] we use local time semantics to increase independence. For timed automata the implicit synchronization of global time restricts independence of actions. For our component model we note that the separation of communication from internal computations makes internal transitions independent of actions in other components. We also note that we have extensive information on how components communicate, which is useful for our ample set construction. To increase independence further we relax the synchronization, so that we abstract from the exact time of

[1] SAVE is a project supported by Swedish Foundation for Strategic Research. See `http://www.mrtc.mdh.se/SAVE/` for more information.

non-triggering write operations while preserving the order of writes. We present an algorithm for partial order reduction that takes advantage of these ideas, and provide some experimental results from a prototype implementation. The experiments indicate that that our component model is well suited for partial order reduction, and they show good additional reductions when the further relaxed synchronization is used.

Related work includes partial order reduction techniques for timed systems, in particular the local time semantics for timed automata first introduced by Bengtsson et. al., [3]. They also apply the partial order reduction in reachability analysis of timed automata. This work is extended to Timed LTL model-checking by Minea in [17]. We adopt the local time semantics to develop a reachability analysis algorithm for a component model and study how the particular semantics and the static structure of the model can be used to improve previous results. A more recent approach to symbolic model checking of timed automata based on partial order semantics is presented by Lugiez et. al., in [15]. It relies on constraints over event occurrences, instead of clock constraints. In [19], Salah et. al., show that the union of zones reached by interleavings of the same set of transitions is convex. Concurrent semantics for networks of timed automata are investigated in [6,4], by a symbolic unfolding into petri nets with read arcs (to support urgency and invariants).

The rest of this paper is organized as follows: the component model is described in Section 2. In Section 3 we describes our approach to partial order reduction, and in Section 4 we give an algorithm for checking reachability and presents results from an experiment. Section 5 concludes the paper.

2 The Component Model

We introduce timed behaviours to model the execution of components as timed automata, and go on to define syntax and semantics for our component model.

Example 1 (Running Example). Figure 1 shows a horizontal composition of components A, B and C. Assume A is a timer, C a controller, and B a component generating setpoint for the controller. The timer A is connected to the input trigger port p_3 to periodically activate C. The port p_5 is used to read sensor input, which is compared to the setpoint when the controller computes its output to the actuator, port p_6.

2.1 Timed Behaviour

We define a timed behaviour as a timed automaton, extended with data variables and a final location such that no edges are leading out from this location. For a timed behaviour we have two sets of variables, the set V_C of clock variables, and V_D of data variables. The domain of variables in V_C is the non-negative real numbers $\mathbb{R}_{\geq 0}$, and for variables in V_D the domain is a bounded set of integers `INT`. We denote by $\mathcal{P}(V_C)$ the power-set of V_C, i.e. the set of all subsets of V_C.

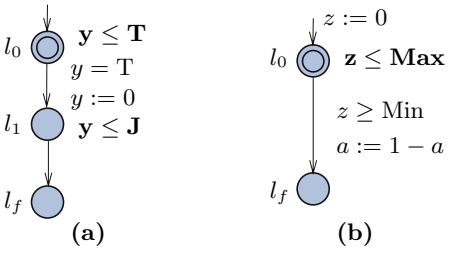


Fig. 2. Timed behaviours: **(a)** a timer with period T and jitter J, **(b)** a computation updating data variable a after between Min and Max time units

A term t is generated by the grammar $t ::= m \mid y \mid t_1 \otimes t_2$, where m is a natural number, $y \in V_\mathrm{D}$ is a data variable, $\otimes \in \{+, -, \times, /\}$, and t_1, t_2 are terms. Let U be the set of *variable updates*, each in the form $y := t$ for a data variable $y \in V_\mathrm{D}$ and a term t. An atomic clock constraint is of the form $y \sim m$, for $y \in V_\mathrm{C}$, m a natural number, and $\sim \in \{<, \leq, =, \geq, >\}$. Similarly, an atomic data constraint is of the form $t_1 \sim t_2$, with terms t_1 and t_2. We denote by $\mathrm{conj}(V_\mathrm{D}, V_\mathrm{C})$ the set of conjunctions of atomic constraints. For $g \in \mathrm{conj}(V_\mathrm{D}, V_\mathrm{C})$ we have $g_\mathrm{D} \in \mathrm{conj}(V_\mathrm{D})$ the atomic data constraints of g, and $g_\mathrm{C} \in \mathrm{conj}(V_\mathrm{C})$ the atomic clock constraints of g.

A *timed behaviour* is a timed automaton $\mathcal{B} = \langle N, l_0, l_f, V_\mathrm{D}, V_\mathrm{C}, r_0, r_f, E, I \rangle$, where N is a set of locations, l_0 is the initial location, l_f is the final location, V_D and V_C are sets of variables, $r_0 \subseteq V_\mathrm{C}$ and $r_f \subseteq V_\mathrm{C}$ are sets of clocks (initial and final resets), E is a set of edges so that $E \subseteq N \times \mathrm{conj}(V_\mathrm{D}, V_\mathrm{C}) \times U \times \mathcal{P}(V_\mathrm{C}) \times N$, and I maps each location l in $N \setminus \{l_f\}$ to its invariant $I(l)$, a conjunction of upper bounds on clocks ($y \leq m$ or $y < m$). We write $l \xrightarrow{g,e,r} l'$ iff $\langle l, g, e, r, l' \rangle \in E$ to denote an edge from location l to l' with a guard g, variable update e, and reset clocks $r \subseteq V_\mathrm{C}$.

2.2 Component

A component in our setting is defined by its interface and a timed behaviour. The interface of a component consists of data ports and trigger ports, where connected data ports define data flow between components, and connected trigger ports define control flow. The ports are either input or output. An input data port has an associated data variable holding the current data item for this port.

A component is initially *idle*, and it remains in this state until all input trigger ports have been activated, at which point it is *triggered* and switches to the executing state. The internal computation of a component starts with a *read* phase, where all the input data ports are stored internally. The internal copies of input data are used together with internal state variables during the *execute* phase, where the internal behaviour of the component is executed. When the execute phase is over the *write* phase writes output to the output data ports. Finally, the input trigger ports are reset and all outgoing trigger ports are activated, after which the component returns to the idle state.

Each component C is a tuple $\langle P_{\text{in}}, P_{\text{out}}, P_{\text{trig}}, \mathcal{B}, \pi \rangle$, where P_{in} is a set of input ports, P_{out} is a set of output ports, $P_{\text{trig}} \subseteq P_{\text{in}}$ are the trigger input ports, $\mathcal{B}$ is the timed behaviour, and $\pi : P \mapsto V_{\text{D}}$ is a mapping from ports to variables. We denote by P the set of ports $P_{\text{in}} \cup P_{\text{out}}$ of a component.

Example 2. Component A of Example 1 has no input ports so it is spontaneously triggered. Figure 2 (a) shows the timed behaviour $\mathcal{B}^{\text{A}}$ of A, with period $\text{T} = 10$ and jitter $\text{J} = 1$. When A is triggered the read phase leads to the initial location l_0. The automaton uses a clock y to ensure that l_f is reached every T time units, with a non-deterministic offset J. Reaching the final location l_f starts the write phase, after which A is spontaneously triggered again. Figure 2 (b) shows the timed behaviour $\mathcal{B}^{\text{B}}$ of component B, with response time between $\text{Min} = 5$ and $\text{Max} = 20$. The clock z is used to ensure that l_f is reached after between Min and Max time units. The setpoint value a is updated to $1 - a$. The port mapping π^{B} for B is such that $\pi^{\text{B}}(p_2) = a$, meaning that a is copied to p_2 in the write phase.

Semantics. In order to define a state of a component we first introduce clock and data valuations. For a set of clocks V_{C} a clock valuation is a map $u : V_{\text{C}} \mapsto \mathbb{R}_{\geq 0}$. Similarly, for a set of data variables V_{D} and ports P a data valuation is a map $v : (V_{\text{D}} \cup P) \mapsto \texttt{INT}$. Operations on valuations are:

$$
\begin{array}{rll}
u' = [r := 0]u & \text{iff} & u'(y) = 0 \text{ for clocks } y \in r, \text{ and} \\
 & & u'(y') = u(y') \text{ for } y' \notin r. \\
v' = [y := t]v & \text{iff} & v'(y) = v(t) \text{ for } y, \text{ and} \\
 & & v'(y') = v(y') \text{ for } y' \neq y. \\
u' = u \oplus \delta & \text{iff} & \delta \in \mathbb{R}_{\geq 0} \text{ and } u'(y) = u(y) + \delta \text{ for any clock } y.
\end{array}
$$

We introduce the *idle* location $l_\perp \notin N$, and denote by $N_\perp$ the set $N \cup \{l_\perp\}$. A *state* of a component is a tuple $\langle l, v, u \rangle$, where l is a location in $N_\perp$, v is a data valuation, and u is a clock valuation. We introduce values active and inactive for trigger ports, and define a component as *triggered* for a data valuation v, $\text{triggered}(v)$, iff for each $p \in P_{\text{trig}}$ we have $v(p) = \text{active}$. A function $\text{input}(v)$ is used to copy values from input ports to corresponding internal variables, similarly $\text{output}(v)$ copies internal variables to output ports, and $\text{idle}(v)$ inactivates trigger inputs:

$$
\begin{array}{ll}
\text{input}(v) & = [y := p \,|\, p \in P_{\text{in}},\ y = \pi(p)]v \\
\text{output}(v) & = [p := y \,|\, p \in P_{\text{out}},\ y = \pi(p)]v \\
\text{idle}(v) & = [p := \text{inactive} \,|\, p \in P_{\text{trig}}]v
\end{array}
$$

The transition rules for a component C are:

- *delay transition*: $\langle l, v, u \rangle \xrightarrow{\delta} \langle l, v, u \oplus \delta \rangle$ if $\delta \in \mathbb{R}_{\geq 0}$, $u \oplus \delta \models I(l)$, $l \neq l_f$, and if $l = l_\perp$ then $\neg\text{triggered}(v)$.
- *internal transition*: $\langle l, v, u \rangle \xrightarrow{\tau} \langle l', v', u' \rangle$ along an edge $l \xrightarrow{g,e,r} l'$ with e in the form $y := t$ if $v \models g_{\text{D}}, u \models g_{\text{C}}$, $u' \models I(l')$, $v' = [y := t]v$, and $u' = [r := 0]u$.

- *read transition*: $\langle l_\perp, v, u\rangle \xrightarrow{r} \langle l_0, \text{input}(v), [r_0 := 0]u\rangle$ if triggered(v).
- *write transition*: $\langle l_f, v, u\rangle \xrightarrow{w} \langle l_\perp, \text{idle}(\text{output}(v)), [r_f := 0]u\rangle$.

We use the following restriction on the internal behaviour of components to avoid spurious local time traces:

Definition 1 (Time Divergence). *We require for a timed behaviour that time diverges, i.e. that there is no time-stop due to invariants, and within any finite time bound only a finite number of transitions can be taken (non-zenoness).*

2.3 Composition

A composition is a set of interconnected components, also known as a horizontal composition. We define a composition as a tuple $\langle P^{\text{x}}_{\text{in}}, P^{\text{x}}_{\text{out}}, \mathcal{C}, \mathcal{X}\rangle$, where P^{x}_{in} and $P^{\text{x}}_{\text{out}}$ are external ports connecting the composition to its environment, $\mathcal{C}$ is a set of components and $\mathcal{X}$ is a set of connections. A connection $x = \langle p, p'\rangle$ connects port $p \in (P^i_{\text{out}} \cup P^{\text{x}}_{\text{in}})$ to port $p' \in (P^j_{\text{in}} \cup P^{\text{x}}_{\text{out}})$ for C^i and C^j in $\mathcal{C}$. We do not allow conflicting connections, i.e. connecting output ports of the same component with the same port.

Example 3. The composition of Fig. 1 has an external input port p_5 for sensor input, an external output port p_6 for actuation, three components A, B and C, and two connections $\langle p_1, p_3\rangle$ and $\langle p_2, p_4\rangle$.

Semantics. A state of a composition is a triple $\langle l, v, u\rangle$, where l is a location vector, v is a data valuation and u is a clock valuation. For a state s we denote by $s[i]$ the state $\langle l[i], v[i], u[i]\rangle$ of a component $C^i \in \mathcal{C}$. The local valuations $v[i]$ and $u[i]$ for a component C^i are such that $v[i](y) = v(y)$ for $y \in (P^i \cup V^i_{\text{D}})$, and $u[i](y) = u(y)$ for $y \in V^i_{\text{C}}$. In addition to the local valuations, the data valuation v also maps external ports P^{x} to their values. The transfer of data and triggering introduced by writing to ports Q:

$$\begin{aligned}\text{writedata}(Q, v) &= [p' := p \,|\, \langle p, p'\rangle \in \mathcal{X}, p \in Q, p' \in P^{\text{x}}_{\text{out}} \cup P^j_{\text{in}} \setminus P^j_{\text{trig}}]v\\ \text{writetrig}(Q, v) &= [p' := \text{active} \,|\, \langle p, p'\rangle \in \mathcal{X}, p \in Q, p' \in P^j_{\text{trig}}]v\end{aligned}$$

The transition rules for a composition are then:

- *delay transition*: $s \xrightarrow{\delta} s'$ if $s[i] \xrightarrow{\delta} s'[i]$ for each component $C^i \in \mathcal{C}$.
- *internal transition*: $s \xrightarrow{\tau^i} s'$ in the behaviour of C^i if $s[i] \xrightarrow{\tau} s'[i]$, and $s[j] = s'[j]$ for $j \neq i$.
- *read transition*: $s \xrightarrow{r^i} s'$ if $s[i] \xrightarrow{r} s'[i]$ and $s'[j] = s[j]$ for $j \neq i$.
- *write transition*: $s \xrightarrow{w^i} s'$ where either $C^i \in \mathcal{C}$ for internal component C^i writing to ports $Q = P^i_{\text{out}}$ or $i = Q$ for external write to ports $Q \subseteq P^{\text{x}}_{\text{in}}$ if
  - internal state of writer is updated: $s[i] \xrightarrow{w} s_1[i]$ if $C^i \in \mathcal{C}$, $s_1[j] = s[j]$ for $j \neq i$ (for external writes $s_1 = s$), and
  - data or triggering is transferred from ports Q: $s' = \langle l_1, v', u_1\rangle$ with $v' = \text{writetrig}(Q, \text{writedata}(Q, v_1))$.

2.4 Composite Component

In our component model [1,5] we introduce composite components to support hierarchical composition, by allowing the behaviour of a component to be described by a composition. As any other component, a composite component is defined by its interface (ports) and its timed behaviour. Unlike other components the behaviour of a composite component is described as an internal composition. This is sometimes referred to as vertical composition. Composite components can be constructed from compositions that are time divergent (Definition 1). We also require that internal components have at least one input trigger port, to avoid spontaneous triggering.

For a composite component C, an internal transition is either an internal, read or write transition of some internal component. The read operation of C correspond to a write to the external input ports of the internal composition. Component C can write when all internal components are idle. The port values are already updated by internal writes, so the write operation only need to inactivate input trigger ports.

3 Partial Order Reduction

The idea of partial order reduction is to explore representative sequences of independent transitions, instead of examining all possible sequences. However, the implicit synchronization of global time restricts independence for transitions of timed automata. As in [3,17] we use local time semantics to increase independence. It essentially allow us to analyse components of a composition in isolation, and then synchronize the components to a shared state whenever one writes to the others. To increase independence further than [3,17] we relax the synchronization, so that we abstract from the exact time of non-triggering write operations.

3.1 Representatives and Local Time Traces

To describe the concept of representative traces, we first need a notion of independent transitions. Two transitions are considered independent if they can be reordered within a trace without affecting the final state of the trace, or the validity of the trace. We denote by enabled(σ) the set of transitions that can immediately follow a finite trace σ, and define independent transitions as in [17]:

Definition 2. *Two transitions α_1 and α_2 are independent iff for any trace σ such that $\alpha_1, \alpha_2 \in \text{enabled}(\sigma)$:*

- Enabledness: $\alpha_2 \in \text{enabled}(\sigma\alpha_1)$ *and* $\alpha_1 \in \text{enabled}(\sigma\alpha_2)$.
- Commutativity: *Any state reachable by the trace $\sigma\alpha_1\alpha_2$ can also be reached by the trace $\sigma\alpha_2\alpha_1$.*

Independence is a sufficient condition for reordering transitions within a trace so that the same state is reached, however it is not a sufficient condition for

reordering of transitions during reachability analysis. There could for example be a transition α_3 in enabled($\sigma\alpha_1$) which is not in enabled($\sigma\alpha_2$), which we would miss if we only considered the trace $\sigma\alpha_2\alpha_1$. For analysis of a composition we need a strategy to make sure that some representative trace is explored for each possible trace of the full state graph. We examine conditions for reordering further in Sect. 4.1.

Independence can be concluded from the structure of a composition, using a set active(α) of components that participate in a transition α such that active(α^i) = $\{C^i\}$ for $\alpha \in \{\tau, \delta, r\}$ and active(w^i) = $\{C^i\} \cup \{C^j \mid \langle p, p' \rangle \in \mathcal{X}, p \in P^i_{\text{out}}, p' \in P^j_{\text{in}}\}$. We then restate a theorem found in e.g. [3,17], i.e. that two local time actions are independent if no automata participate in both actions:

Theorem 1. $\text{active}(\alpha_1) \cap \text{active}(\alpha_2) = \emptyset \;\Rightarrow\; \text{independent}(\alpha_1, \alpha_2)$

Proof. See [17]. □

We reduce the independence relation further (for internal transitions) in Sect. 3.2, where we introduce local time semantics. We say that two transitions α_1 and α_2 are dependent, and write dependent(α_1, α_2), whenever independent(α_1, α_2) cannot be concluded from the structure of a composition. The dependency relation is thus a safe approximation of transitions that are not independent.

We define a local time trace to be a representative of some timed trace. A trace σ is a representative of a trace σ' iff independent transitions of σ can be reordered to construct σ'. A timed trace (as defined in [2]) is a pair $\langle \sigma, t \rangle$ such that $\sigma(i)$ is the i*th* transition of the trace, $t(i)$ is the time of this transition, and the timepoints $t(i)$ are monotonic, i.e. $i \leq j$ implies $t(i) \leq t(j)$. Local time traces are then defined as:

Definition 3. *A* local time trace *is a trace $\langle \sigma, t \rangle$ such that dependent transitions are monotonic, i.e. for any $\sigma(i)$ and $\sigma(j)$ that are dependent we have $i \leq j$ implies $t(i) \leq t(j)$.*

3.2 Local Time Semantics

To keep track of the local time within components we introduce a reference clock $c^i \in V^i_{\text{C}}$ for each component $C^i \in \mathcal{C}$. We define a local delay transition for component C^i, where other components do not need to delay correspondingly.

We relax the synchronization of our local time semantics compared to [3] by completing the internal computation of a component before synchronizing with other components. This can be done since values written to input ports of a component are not used during its internal computation. We run internal computations ahead implicitly in the rule for a write operation w^i of C^i by requiring that any C^j such that dependent(w^i, w^j) is either idle ($l[j] = l^j_\perp$) or finished ($l[j] = l^j_f$).

Time is synchronized so that the local time in all components dependent on a write operation is ensured to be later than the local time of the writer, i.e.

$c^i \leq c^j$ for a writer C^i and any C^j such that dependent(w^i, w^j). This is to make traces of the local time semantics be *local time traces*, according to Definition 3. The set of components that are triggered by a write, given a location vector l and data valuation v, is:

$$\text{trig}(l, v) \;=\; \{C^j \mid l[j] = l^j_\perp \wedge \text{triggered}^j(v[j])\}$$

The clock synchronization constraint $\text{sync}^i(l, v)$ also preserves the exact time of triggering by requiring that $c^i = c^j$ for triggered components C^j:

$$\text{sync}^i(l, v) \quad = \quad \left(\bigwedge_{C^j \in \text{trig}(l,v)} c^i = c^j \right) \wedge \left(\bigwedge_{\text{dependent}(w^i, w^j)} c^i \leq c^j \right)$$

We use compositions to describe component based systems. We define local time transitions $s \longrightarrow_t s'$ for a composition using the corresponding rules for transitions $s \longrightarrow s'$:

- *delay transition*: $s \xrightarrow{\delta^i}_t s'$ if $s[i] \xrightarrow{\delta} s'[i]$ and $s'[j] = s[j]$ for $j \neq i$.
- *internal transition*: $s \xrightarrow{\tau^i}_t s'$ if $s \xrightarrow{\tau^i} s'$.
- *read transition*: $s \xrightarrow{r^i}_t s'$ if $s \xrightarrow{r^i} s'$.
- *write transition*: $s \xrightarrow{w^i}_t s'$ for $s = \langle l, v, u\rangle$ if $s \xrightarrow{w^i} s'$, $u \models \text{sync}^i(l, v)$, and either $l[j] = l^j_\perp$ or $l[j] = l^j_f$ for C^j such that dependent(w^i, w^j).

Lemma 1 shows that internal transitions in the local time semantics are independent of write transitions. This independence makes our model suited for partial order reduction, and enables the weak synchronization (otherwise we would need $c^i = c^j$ also for non-triggering participants, instead we use $c^i \leq c^j$ to preserve order of write operations).

Lemma 1. *For the local time semantics we have:*

- independent(α^i_1, α^j_2) *if* $i \neq j$ *and* α_1, α_2 *in* $\{\tau, \delta\}$
- independent(w^i, α^j) *if* $i \neq j$ *and* α *in* $\{\tau, \delta\}$
- independent(w^i, r^j) *if* $C^j \notin \text{active}(w^i)$
- independent(w^i, w^j) *if* $\text{active}(w^i) \cap \text{active}(w^j) = \emptyset$

Proof. by Theorem 1, and for independent(w^i, τ^j) we note that w^i is not enabled if τ^j is enabled for $C^j \in \text{active}(w^i)$, similarly for δ^j. □

Theorem 2 (Correctness of Local Time Semantics). *Assume local time states* s_0 *and* $s_f = \langle l_f, v_f, u_f\rangle$, *global time states* s'_0 *and* $s'_f = \langle l'_f, v'_f, u'_f\rangle$, *and a component* C^k. *Let* $s_0 = s'_0$ *except for reference clocks which are zero in* s_0 *but not included in* s'_0.

- *(*Soundness*) whenever* $s_0 \longrightarrow^*_t s_f$ *then* $s'_0 \longrightarrow^* s'_f$ *so that* $l_f[k] = l'_f[k]$, $v_f[k](y) = v'_f[k](y)$ *for* $y \in V^k_{\text{D}}$ *(i.e. not for* $y \in P^k$*), and* $u_f[k] = u'_f[k]$.
- *(*Completeness*) whenever* $s'_0 \longrightarrow^* s'_f$ *then* $s_0 \longrightarrow^*_t s_f$ *so that* $l_f[k] = l'_f[k]$, $v_f[k](y) = v'_f[k](y)$ *for* $y \in V^k_{\text{D}}$ *(i.e. not for* $y \in P^k$*), and* $u_f[k] = u'_f[k]$.

Proof. Soundness is shown by induction over a local time trace, and completeness by construction of the corresponding local time trace (see [12]). □

3.3 Symbolic Local Time Semantics

We define a zone Z as a set of clock valuations, and use symbolic states $\langle l, v, Z\rangle$ to represent all states $\langle l, v, u\rangle$ such that $u \in Z$. For a zone Z and a clock constraint g we define *conjunction* $Z \wedge g$ as the set of valuations $u \in Z$ such that $u \models g$, *reset* $r(Z)$ as u' such that $u' = [r := 0]u$ for $u \in Z$, and *symbolic local delay* $Z^{\uparrow i}$ as u' such that for $u \in Z$, $\delta \in \mathbb{R}_{\geq 0}$ we have $u'[i] = u[i] \oplus \delta$ and $u'[j] = u[j]$ for $i \neq j$.

The initial symbolic state is $\langle l_0, v_0, Z_0\rangle$, where $Z_0 = (\{u_0\})^{\uparrow} \wedge I(l_0)$ incorporates an initial delay in all components from a zone with a single solution u_0. To improve the presentation we incorporate the semantics of a component into the transition rules for a composition:

- *internal transition*: $\langle l, v, Z\rangle \stackrel{\tau^i}{\Longrightarrow}_t \langle [l^{i\prime}/l^i]l, v', Z'\rangle$ along an edge $l^i \stackrel{g,e,r}{\longrightarrow} l^{i\prime}$ with e in the form $y := t$ if $v \models g_{\mathrm{D}}$, $v' = [y := t]v$, and
  if $l^{i\prime} = l^i_f$ then $Z' = r(Z \wedge g_{\mathrm{C}})$, otherwise $Z' = (r(Z \wedge g_{\mathrm{C}}))^{\uparrow i} \wedge I^i(l^{i\prime})$.
- *read transition*: $\langle l, v, Z\rangle \stackrel{r^i}{\Longrightarrow}_t \langle [l^i_0/l^i_{\perp}]l, v', Z'\rangle$ if $l[i] = l^i_{\perp}$ and $\mathrm{triggered}^i(v[i])$, with $v'[i] = \mathrm{input}(v[i])$, $v'[j] = v[j]$ for $j \neq i$, and
  if $l^i_0 = l^i_f$ then $Z' = r^i_0(Z)$, otherwise $Z' = (r^i_0(Z))^{\uparrow i} \wedge I^i(l^i_0)$.
- *write transition*: $\langle l, v, Z\rangle \stackrel{w^i}{\Longrightarrow}_t \langle [l^i_{\perp}/l^i_f]l, v', Z'\rangle$ where either $C^i \in \mathcal{C}$ writing to ports $Q = P^i_{\mathrm{out}}$, or $i = Q$ for external write to $Q \subseteq P^{\mathrm{x}}_{\mathrm{in}}$, if $l[i] = l^i_f$ and:
  - $l[j] = l^j_{\perp}$ or $l[j] = l^j_f$ for C^j such that $\mathrm{dependent}(w^i, w^j)$,
  - $v_1[i] = \mathrm{idle}(\mathrm{output}(v[i]))$ if $C^i \in \mathcal{C}$, $v_1[j] = v[j]$ for $j \neq i$,
  - $v' = \mathrm{writetrig}(Q, \mathrm{writedata}(Q, v_1))$, and
  - if $\mathrm{triggered}^i(v'[i])$ then $Z' = r^i_f(Z \wedge \mathrm{sync}^i(l, v))$, otherwise $Z' = (r^i_f(Z \wedge \mathrm{sync}^i(l, v)))^{\uparrow i}$.

For global time semantics a zone can be represented as a conjunction of clock difference constraints. Constraints on two clocks are preserved by global time delay because both clocks grow equally, but for local time we need a different zone representation that is preserved by local time delay. Local time zones can be efficiently represented [3,17] as difference constraints on reference clocks c^i and timestamps t_y for the latest reset of a clock y.

Example 4. We explore a symbolic trace of the composition in Fig. 1. In the initial state $\langle [l^{\mathrm{A}}_{\perp}, l^{\mathrm{B}}_{\perp}, l^{\mathrm{C}}_{\perp}], v_0, Z_0\rangle$ both A and B can read, since they have no trigger input ports (and so all their triggers are trivially active). If A reads first (r^{A}) we get to a state $\langle [l^{\mathrm{A}}_0, l^{\mathrm{B}}_{\perp}, l^{\mathrm{C}}_{\perp}], v_0, Z_1\rangle$ with $Z_1 = Z_0^{\uparrow \mathrm{A}} \wedge (y \leq 10)$. We continue the trace with r^{B} to $\langle [l^{\mathrm{A}}_0, l^{\mathrm{B}}_0, l^{\mathrm{C}}_{\perp}], v_0, Z_2\rangle$ with $Z_2 = r_z(Z_1)^{\uparrow \mathrm{B}} \wedge (z \leq 20)$ for $r_z = \{z\}$. From this state internal transitions τ_A and τ_B are enabled. By τ^{B} we get to a state $\langle [l^{\mathrm{A}}_0, l^{\mathrm{B}}_f, l^{\mathrm{C}}_{\perp}], v_3, Z_3\rangle$ with $v_3 = [a := 1]v_0$ and $Z_3 = Z_2 \wedge (z \geq 5)$. Component B cannot write from this state, because w^{B} depends on w^{A} and A is neither idle or in its final location, so we take τ^A to the state $\langle [l^{\mathrm{A}}_1, l^B_f, l^C_{\perp}], v_3, Z_4\rangle$ with $Z_4 = r_y(Z_3 \wedge y = 10)^{\uparrow \mathrm{A}} \wedge y \leq 1$ for $r_y = \{y\}$. Another τ^{A} leads to $\langle [l^{\mathrm{A}}_f, l^{\mathrm{B}}_f, l^{\mathrm{C}}_{\perp}], v_3, Z_4\rangle$.

From this state both A and B can write, A when $10 \leq c^{\mathrm{A}} \leq 11$ and B when $5 \leq c^{\mathrm{B}} \leq 20$, but it cannot be determined if C will get data from B before being triggered by A.

Theorem 3 (Correctness of Symbolic Local Time Semantics). *Assume location vectors l_0, l_f, variable valuations v_0, v_f, clock valuations u_0, u_f, and local time zones Z_0, Z_f.*

- *(*Soundness*) whenever* $\langle l_0, v_0, Z_0\rangle \Longrightarrow_t^* \langle l_f, v_f, Z_f\rangle$ *then for any* $u_f \in Z_f$ $\langle l_0, v_0, u_0\rangle \longrightarrow_t^* \langle l_f, v_f, u_f\rangle$.
- *(*Completeness*) whenever* $\langle l_0, v_0, u_0\rangle \longrightarrow_t^* \langle l_f, v_f, u_f\rangle$ *then* $\langle l_0, v_0, Z_0\rangle \Longrightarrow_t^* \langle l_f, v_f, Z_f\rangle$ *so that* $u_f \in Z_f$.

Proof. Symbolic transition rules are constructed from local time semantics and definitions of zone operations. Preservation of zone representation is shown in [17]. See [12]. □

4 Reachability Analysis

We perform reachability analysis by exploring a subset of enabled transitions from each explored state, in order to reach a target location denoted $l_{\odot}^{k}$ for a component C^k. In the analysis we use the symbolic local time semantics, to get the independence introduced in our local time semantics and to get a finite state space.

4.1 The Ample Set Method

An ample set [18] is a subset of the enabled transitions that is sufficient to explore when model checking. The ample set method reduces a state graph G to a subgraph R such that correctness of model checking is preserved, i.e. checking R gives the same result as checking G. In general we need to select an ample set so that the checked property is preserved, i.e. a property holds in a representative trace σ in R iff the same property holds in all traces σ' in G represented by σ, and so that all traces in G have a representative in R.

For local reachability we note first that we only need to consider traces of G that can actually reach the target location. We also note that local reachability in a component C^k is preserved by representative traces, since the reordering of independent transitions does not affect which local states are reachable. The following describes how we can construct an ample set:

Definition 4 (Ample Set Construction). *From the static structure of a composition we compute $\mathcal{P}^k$ the enabled transitions that can give progress in C^k, i.e. transitions of C^k or of some C^j producing data or triggering to C^k (possibly via other components). From $\mathcal{P}^k$ we define the ample set as follows:*

- *The ample set is empty iff $\mathcal{P}^k$ is empty, otherwise some $\alpha_0 \in \mathcal{P}^k$ is in the ample set. This must hold for each valuation of a zone, as discussed in [17].*

- *If α is in the ample set, α' is enabled, and* dependent(α, α') *then α' must also be in the ample set.*
- *If α is in the ample set, α' is not enabled,* dependent(α, α')*, and there is a transition α'' that can lead to $\alpha' \in \mathcal{P}^k$ before α is taken, then α'' must also be in the ample set. This can be determined from the static structure of a composition.*
- *For a local cycle, at least one transition in $\mathcal{P}^k$ and outside of the cycle is in the ample set.*

Theorem 4. *For any trace σ in G reaching the target location, the subgraph R induced by the ample set construction in Definition 4 contains a representative of an extension $\sigma\rho$ in G.*

Proof. By a construction similar to that in [7], with simplifications for local reachability, and cycle closing due partly to construction of $\mathcal{P}^k$ (see [12]). □

As mentioned in [18] constructing an optimal ample set with respect to state space reduction is NP hard, so we suggest a heuristic. The construction in Definition 4 starts from a transition α_0 in $\mathcal{P}^k$, according to the first rule. Once a transition has been selected the other rules are used to find the least fixpoint, which is an ample set. To reduce the size of the ample set we select α_0 as an internal transition τ^i, if possible. Otherwise select α^i with minimal upper bound on the reference clock of component C^i. This reduces the possibility for other components to interfere with the execution of α^i. We also prefer read operations over writes when selecting α_0.

4.2 Model Checking Algorithm

An algorithm for symbolic reachability analysis based on the symbolic local time semantics and ample set construction is shown in Fig. 3. It is a standard reachability algorithm, with the exception that the *ample* transitions are explored instead of all enabled transitions. The normalisation $\max(Z)$ is used to ensure termination, as the symbolic semantics is not finite. In [3] it is shown that there is a finite partitioning of the state space, and [17] suggests a method for constructing $\max(Z)$. Efficient representations of local time zones are also discussed in [3,17].

We expect the algorithm to perform well: the partial order reduction is a subset of the symbolic local time semantics, by exploring only the ample set: a subset of enabled transitions. The local time semantics is sound (Theorem 2) with respect to the global time semantics. The partitioning of the state space induced by $\max(Z)$ is however incomparable with the normalisation of global time zones, so we cannot conclude any strict improvements. For timed automata with local time semantics [3,17] soundness is shown only for synchronized states, and in general for a local time state there might not exist a corresponding synchronized state. This means that some local time traces lead outside the global time semantics, giving a larger state graph to search. The components of our model are time divergent, can always accept input, and never require input to

PASSED := ∅
WAITING := $\{\langle l_0, v_0, Z_0\rangle\}$
repeat for some $\langle l, v, Z\rangle \in$ WAITING
  WAITING := WAITING $\setminus \{\langle l, v, Z\rangle\}$
  if $l[k] = l^k_\odot$ **then return "REACHABLE"**
  else if $\max(Z) \not\subseteq Z'$ **for all** $\langle l, v, Z'\rangle \in$ PASSED **then**
    PASSED := PASSED $\cup \langle l, v, \max(Z)\rangle$
    SUCC := $\{\langle l', v', Z'\rangle \mid \langle l, v, Z\rangle \stackrel{\alpha}{\Longrightarrow}_t \langle l', v', Z'\rangle,\ \alpha \in \text{ample}(l, v, Z)\}$
    WAITING := WAITING $\cup$ SUCC
until WAITING = ∅
return "NOT REACHABLE"

Fig. 3. An algorithm for symbolic reachability analysis, exploring a selected subset of enabled transitions from each state until component C^k reaches $l^k_\odot$

be available. Because of this components can always catch up, which is why we can show soundness for local states.

Example 5. A symbolic trace is described in Example 4. The local traces for A and B are $r^{\mathrm{A}}\tau^{\mathrm{A}}\tau^{\mathrm{A}}$ and $r^{\mathrm{B}}\tau^{\mathrm{B}}$, respectively. For global time semantics there are six possible interleavings, although probably more interesting nine states are passed. When searching for a location in C using the ample set construction in Definition 4 we select either r^{A} and r^{B} first. The selected component will reach its final location before the other components read operation is selected. Even though we have not specified which transition to select only one trace is explored, because r^{A} and r^{B} are independent. The trace explored using the ample set construction passes through six states, instead of the nine for global time semantics.

4.3 Experimental Results

We have developed a prototype implementation of our method as an extension of the UPPAAL tool[2]. Figure 4 illustrates the synthetic benchmarks we use to evaluate our implementation (similar to the benchmarks of Salah et.al. [19]). A synthetic benchmark NxM is a grid of components in N columns and M rows. Each component $C_{n,m}$ is connected to $C_{n,m+1}$ and to $C_{n+1,m+1}$. The components in the first row are triggered once, and the timed behaviour of each component is a delay by at least 4 time units. As target component C^k for our ample set construction we use $C_{N,M}$ (i.e. $C_{5,3}$ for Fig. 4). Table 1 shows the computation time and number of explored states for *global* time semantics (Section 2), local time with *strict* synchronization (using constraints $c^i = c^j$ also for w^i, w^j dependent) and *local* time semantics (Section 3). The prototype does not implement the normalisation step $\max(Z)$. Normalisation is not required for the benchmark models, and is turned off in the global time implementation. We note that the algorithms with partial order reduction performs much better than

[2] See the web site `www.uppaal.com` for more information.

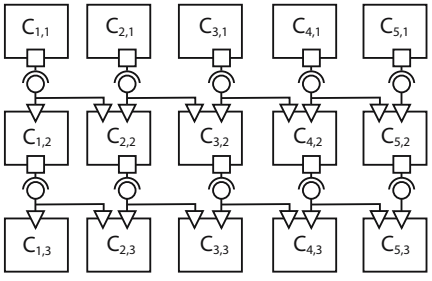


Fig. 4. The synthetic benchmark 5x3 (with N=5 columns, M=3 rows)

Table 1. Benchmark results: we use $\perp$ to denote that the experiment did not terminate in 30 minutes

Method	3x2	3x3	3x4	4x3	4x4
global	264/0.16s	499/0.22s	838/0.31s	2553/0.65s	4778/1.6s
strict	19/0.14s	105/0.19s	443/0.47s	105/0.26s	990/1.7s
local	19/0.14s	65/0.18s	275/0.43s	93/0.25s	336/1.0s
	4x5	5x4	5x5	6x5	6x6
global	8146/3.8s	26108/12s	48096/36s	481318/11m16s	$\perp$
strict	7549/21s	990/3.5s	15892/33s	15892/2m59s	$\perp$
local	2380/10s	598/2.8s	2156/20s	4316/1m08s	15101/7m36s

the algorithm without partial order reduction, and that weak synchronization performs better than the strict. We also note that *global* cover more states per second, this is because of the overheads when synchronizing components and constructing ample sets.

5 Conclusion

In this paper, we have applied and improved existing symbolic partial order reduction techniques for timed automata to develop an efficient model-checking technique for real-time components. The behavior of components are internally described as timed automata that can be hierarchical in the sense that a component can be described as a composition of components. To compose components explicit data and control flow is modeled, a property that is exploited in order to increase the independence between components, and thus to reduce the growth of the state-space caused by interleavings. We give a concrete and symbolic local time semantics for the component model, as well as a symbolic reachability analysis algorithm that uses an ample set construction to select symbolic transitions to explore. We also describe a heuristics that can be used for accelerating the analysis of local reachability properties (e.g., reachability of a location in a single component).

Our experiments suggest that our technique can achieve substantial reduction in the time and space needed to analyze a real-time system described in the studied component model. As future work we plan to further evaluate the reduction in a case study for the component model. We will also complete our implementation of the proposed reachability analysis and evaluate the achieved reduction with respect to existing techniques, such as the *event zones* of [15]. We also plan to further enrich the component model with more complex interaction structures, and support for modeling of other non-functional properties than real-time.

References

1. Åkerholm, M., Carlson, J., Fredriksson, J., Hansson, H., Håkansson, J., Möller, A., Pettersson, P., Tivoli, M.: The SAVE approach to component-based development of vehicular systems. Journal of Systems and Software (2006)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Bengtsson, J., Jonsson, B., Lilius, J., Yi, W.: Partial order reductions for timed systems. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 485–500. Springer, Heidelberg (1998)
4. Bouyer, P., Haddad, S., Reynier, P.-A.: Timed unfoldings for networks of timed automata. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 292–306. Springer, Heidelberg (2006)
5. Carlson, J., Håkansson, J., Pettersson, P.: SaveCCM: An analysable component model for real-time systems. In: Proceedings of the 2nd Workshop on Formal Aspects of Components Software (FACS 2005). Electronic Notes in Theoretical Computer Science, Elsevier, Amsterdam (2005)
6. Cassez, F., Chatain, T., Jard, C.: Symbolic unfoldings for networks of timed automata. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 307–321. Springer, Heidelberg (2006)
7. Clarke, E.M., Grumberg, O., Minea, M., Peled, D.: State space reduction using partial order techniques. International Journal on Software Tools for Technology Transfer 2(3), 279–287 (1999)
8. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering, pp. 109–120. ACM Press, New York (2001)
9. de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Timed interfaces. In: Proceedings of the Second International. Workshop on Embedded Software. LNCS, Springer, Heidelberg (2002)
10. Godefroid, P., Wolper, P.: Using partial orders for the efficient verification of deadlock freedom and safety properties. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 332–342. Springer, Heidelberg (1992)
11. Gössler, G., Sifakis, J.: Composition for component-based modelling. Science of Computer Programming 55(1-3), 161–183 (2005)
12. Håkansson, J., Pettersson, P.: Partial order reduction for verification of real-time components. Technical report, Department of Information Technology, Uppsala University (July 2007)
13. Holzmann, G.: The Design and Validation of Computer Protocols. Prentice Hall, Englewood Cliffs (1991)

14. IEC. International standard IEC 1131: Programmable controllers (1992)
15. Lugiez, D., Niebert, P., Zennou, S.: A partial order semantics approach to the clock explosion problem of timed automata. Theoretical Computer Science 345(1), 27–59 (2005)
16. Lundbäck, K.-L., Lundbäck, J., Lindberg, M.: Development of dependable real-time applications. Arcticus Systems (December 2004)
17. Minea, M.: Partial order reduction for model checking of timed automata. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 431–446. Springer, Heidelberg (1999)
18. Peled, D.: All from one, one for all: on model checking using representatives. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 409–423. Springer, Heidelberg (1993)
19. Salah, R.B., 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)
20. Valmari, A.: A stubborn attack on state explosion. In: Rozenberg, G. (ed.) Advances in Petri Nets 1990. LNCS, vol. 483, Springer, Heidelberg (1991)
21. van Ommering, R., van der Linden, F., Kramer, J., Magee, J.: The Koala component model for consumer electronics software. IEEE Computer, pp. 78–85 (2000)

Guided Controller Synthesis for Climate Controller Using UPPAAL TIGA

Jan J. Jessen[1], Jacob I. Rasmussen[2], Kim G. Larsen[2], and Alexandre David[2]

[1] Automation and Control, Aalborg University, Denmark
jjj@control.aau.dk
[2] Department of Computer Science, Aalborg University, Denmark
illum@cs.aau.dk

Abstract. We present a complete tool chain for automatic controller synthesis using UPPAAL TIGA and Simulink. The tool chain is explored using an industrial case study for climate control in a pig stable. The problem is modeled as a game, and we use UPPAAL TIGA to automatically synthesize safe strategies that are transformed for input to Simulink, which is used to run simulations on the controller and generate code that can be executed in an actual pig stable provided by industrial partner Skov A/S. The model allows for guiding the synthesis process and generate different strategies that are compared through simulations.

1 Introduction

Inevitable parts in a traditional control design cycle are modelling, simulations and synthesis. Modelling often results in non-linear continuous models needing linearization and/or model order reduction in order to be applicable for control, while simulation can implement both original and linearized models. For control synthesis standard linear controllers are verified by design, but the control engineer still needs to perform the step of translating a mathematical description of the controller into an executable application that can be run on an embedded platform. Additionally, in the setting of hybrid models controller synthesis itself is a highly non-trivial task.

In this paper, we present a prototype for model-based framework for optimal control using the recently developed controller synthesis tool UPPAAL TIGA [3,2] in combination with Simulink [10] and Real-Time Workshop [12] providing a complete tool chain for modeling, synthesis, simulation and automatic generation of production code (see Fig. 1). The framework requires that two models of the control problem are provided: An abstract model in terms of a timed game and a complete, dynamic model in terms of a (non-linear) hybrid system. Given the abstract (timed game) model together with logically formulated control and guiding objectives, UPPAAL TIGA automatically synthesizes a strategy which is directly compiled into an S-function[1] representation of the controller. Now

[1] S-function is a term used in Simulink for executable content that can be embedded into Simulink components. S-functions support multiple languages such as C and Matlab.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 227–240, 2007.

using Simulink together with the concrete (dynamic) model, simulation results for additional quantitative aspects of the synthesized controller can be obtained. Alternatively, given interface code for the specific actuators and sensors, Real-Time Workshop allows for the generation of production code implementing the synthesized controller. The glue used to tie UPPAAL TIGA together with Simulink has been hand-coded for the purpose of this paper. For an of the shelf tool chain, this glue need to be implemented into UPPAAL TIGA making S-functions an output format.

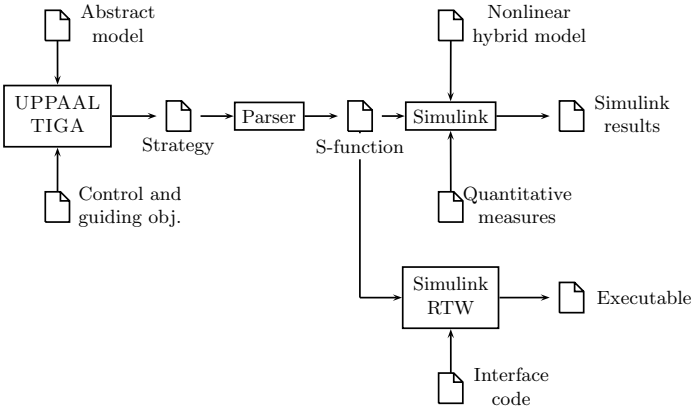


Fig. 1. Illustration of tool chain for model based control

The framework is presented through an industrial case study carried out in collaboration with the company Skov A/S specializing in climate control systems used for modern intensive animal production. For such systems it is of extreme importance that the climate control work properly, since a failure can result in the death of entire batches of animals and in turn loss of revenue for the farmer. In this context a *properly* functioning control system should additionally provide a comfortable environment for the animals.

In [9,8], a dynamic model for a pig stable that is both nonlinear and hybrid and a verified stable temperature controller has been presented. The control design of said papers is unique and does not apply standard control design techniques. We show in this paper that our framework allows for automatic generation of the controller presented in [9,8], and moreover that our framework makes it straight forward to obtain and implement extended controllers, e.g. by including humidity control. For a thorough discussion of the control engineering issues with controller synthesis for the climate controller, we refer to [9,8].

The model of the climate controller is constructed in an ad-hoc manner and the paper does not provide methodology for abstracting non-linear hybrid models to timed automata models. The model serves two different purposes, namely, to illustrate the tool chain for deriving production code from models and provide the area of automatic controller synthesis of real-time systems with an industrial scale case study. Furthermore, the constructed model does not include clock

variables, hence, it does not need the features timed games. However, the climate problem needs a real-time controller and other models using clocks could be constructed, and for that we reason present the work in terms of timed automata. Also, the advanced modelling features of Uppaal Tiga (e.g. functions and selections) make it an attractive choice for modelling games, even if these do not use clock variables.

In Section 2 we describe a dynamic, zone-based climate model for the evolution of temperature in a pig stable. In Section 3 we briefly describe Uppaal Tiga together with the notions of timed game, control objective and strategies. Section 4 is the main section giving a detailed description of how the climate controller is modelled and synthesized with Uppaal Tiga. Numerical results are presented in Section 5, and conclusions are given in Section 6.

2 Climate Model

In this section, we introduce the dynamic climate model describing the evolution of temperature in a pig stable. The presented model is zone based, a concept where the pig stable is divided into distinct climatic zones, and where the zones interact by exchanging air flow. The idea is illustrated in Fig. 2 where a stable is partitioned into N subareas, and where the zones exchange air flow.

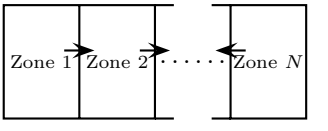


Fig. 2. N zones

Though it would be relevant to model temperature, humidity, CO2 and ammonia concentration we initially limit ourselves to modeling only temperature, in order to illustrate the zone concept. It would though be easy to include the disregarded climate parameters since the mixing dynamics are, roughly, identical.

Assumption 1. *Climatic interdependence between zones is assumed solely through internal air flow.*

With assumption 1 we thus neglect radiation and diffusion etc. between zones, claiming they are negligible compared to the effect from having internal air flow. Besides internal air flow a zone interact with the ambient environment by activating a ventilator in an exhaust pipe and consequently opening a screen to let fresh air into the building. Air flowing from outside into the i^{th} zone is denoted Q_i^{in} [m^3/s], from inside to outside Q_i^{fan} [m^3/s]. Air flowing from zone i to $i+1$ is denoted

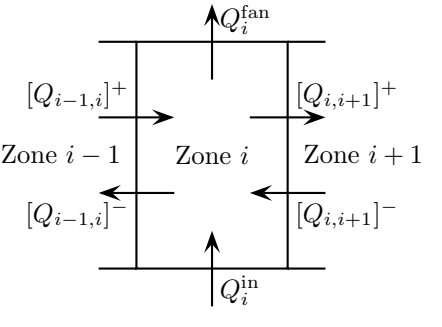


Fig. 3. Illustration of flows for zone i

$Q_{i,i+1}$ [m^3/s] (air flow is defined positive from a lower index to a higher index). A stationary flow balance for each zone i is found:

$$Q_{i-1,i} + Q_i^{\text{in}} = Q_{i,i+1} + Q_i^{\text{fan}} \tag{1}$$

where by definition $Q_{0,1} = Q_{N,N+1} = 0$.

The flow balance is illustrated in Fig. 3 using the following definitions: $[x]^+ \triangleq \max(0, x)$, $[x]^- \triangleq \min(0, x)$. In accordance with [4,1] and taking into account the flows leaving/entering the i^{th} zone, the following model for temperature evolution is easily obtained.

$$\begin{aligned}\frac{dT_i}{dt}V_i = {} & T^{\text{amb}}Q_i^{\text{in}} - T_i Q_i^{\text{fan}} + [Q_{i-1,i}]^+ T_{i-1} - [Q_{i-1,i}]^- T_i \\ & - [Q_{i,i+1}]^+ T_i + [Q_{i,i+1}]^- T_{i+1} + \frac{u_i^t + W_i^t}{\rho_{\text{air}} c_{\text{air}}}\end{aligned} \tag{2}$$

where V_i [m^3] is the zone volume, T^{amb} [$^\circ$C] is the ambient temperature, Q_i^{in}, Q_i^{fan} is the inflow and outflow respectively. c_{air} [J/(kg$^\circ$C)] is the specific heat capacity of air, ρ_{air} [kg/m^3] is the air density. u_i^t [J/s] is the controlled heating and W_i^t [J/s] is heat production from the pigs. For the actuator signals maximum values exists $Q_i^{\text{fan}} \in [0, Q_i^{\text{fan},M}]$, $Q_i^{\text{in}} \in [0, Q_i^{\text{in},M}]$, $u_i^t \in [0, u_i^{t,M}]$. The disturbance is not known but bounded $W_i^t \in [W_i^{t,m}, W_i^{t,M}]$.

In [9] a temperature controller for the model in (2) is presented. The presented controller is a multi-zone controller, i.e., it consists of N individual (yet identical) controllers. The controller is event-based, and only changes its control action when certain boundaries are met or a neighboring zone changes its control action. The controller in [9] is designed to solve a two player game theoretic problem following [7] at each time a state has changed or a change in coordinating variables take place. Each controller maintains a set of coordinating variables δ^i that holds information about the controllers willingness to exchange air flow with the neighboring zones, and only if two neighboring zones agree to the exchange, air will flow between the zones. The game theoretic view of the control problem for an arbitrary zone enables the same generated controller to be implemented in all zones of a N-zone stable. The correctness of this is explicitly proved in [9].

The control actions available to controller is the heating u_i^t, opening of the inlets Q_i^{in} and turning on the ventilators Q_i^{in}. The controller has two "modes" heating up and cooling down, and an initial mode set to either one. We remark specifically that opening of the inlet is not enough to force air into the zone. This being a physical system air has to be removed either by operating the ventilator or by having a neighboring zone extract air. The controller operation in zone i is as follows: When heating up ventilation is closed and heating is turned on. If in addition a neighbor zone has warmer air than in the current zone, the controller will inform the neighboring zone's controller that it would like to receive the warmer air. Only if the two zones agree to exchange air will the controller in zone i turn on its ventilation fan extracting warm air from one of the neighbor zones. When cooling down, the heating is turned off, the inlets opened and

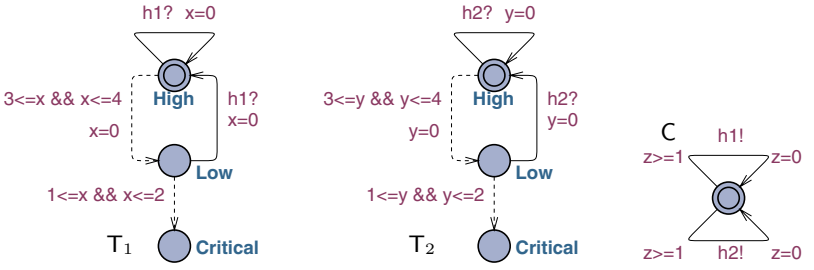


Fig. 4. Two Tank Temperature Control Problem

the ventilation fan is turned on. The controller will in addition inform the two neighbor zones, that it would like to "give away" air thus forcing more fresh air into the zone.

3 Timed Games, Control Objectives and Strategies

Uppaal Tiga is a tool for solving control problems modeled as (networks of) timed game automata [3]. As an example consider the control problem in Fig. 4, where a central controller C is to maintain the temperature of two tanks, T_1 and T_2 above some critical minimum level, say 5°C. Each tank is modelled as a timed game automaton with location `High` indicating that the temperature in the tank is between 80°C and 100°C. Similarly, the `Low` locations indicate a temperature between 10°C and 15°C and the `Critical` locations that temperature is below 5°C. The controller C has the possibility for heating either tank thus lifting (or maintaining) its temperature to the `High` level; the act of heating is modelled as synchronizations on the channels h_1 and h_2. The guards $z \geq 1$ on the clock `z` of the controller enforces that heating actions of C are seperated by at least 1 time-unit. The dashed edges in the two tanks represent *uncontrollable* transitions for lowering the temperature (from `High` to `Low` and from `Low` to `Critical`) in a tank in case no heating action of the controller has taken place for a certain time period; e.g. the guard $3 \leq x \wedge x \leq 4$ indicates that the temperature in T_1 may drop from `High` to `Low` at any moment between 3 and 4 time-units since the last heating of the tank.

Control purposes are formulated as "`control: P`", where `P` is a TCTL formula specifying either a safety property, (`A[]` φ) or a liveness property (`A<>` φ). Given a control purpose, "`control: P`", the search engine of Uppaal Tiga will provide a *strategy* (if any such exists) for the controller under which the behaviour of the model will satisfy `P`. Here a strategy is a function that in any given state of the game informs the controller what to do either in terms of "performing a controllable action" or to "delay". In our tank example of Fig. 4 the control purpose may be formulated as "`control: A[] not(`T_1`.Critical or `T_2`.Critical)`". Endeed there is a strategy guaranteeing the safety property involved (i.e., the `Critical` temperature level is avoided in both tanks). In the case when the two

tanks are both having a `Low` temperature level the strategy provided by UPPAAL TIGA requests the controller to heat T_2 whenever $(2 < \mathtt{y} \wedge 1 < \mathtt{z} \wedge \mathtt{y} \leq \mathtt{x}) \vee ((2 < \mathtt{x} \wedge 1 < \mathtt{z}) \wedge (\mathtt{y} < 1 \vee \mathtt{x} < \mathtt{y}))$. In case $(2 < \mathtt{y} \wedge 1 < \mathtt{z} \wedge \mathtt{x} < 1)$ the strategy suggest to heat T_1. Interestingly, it may be shown (as discovered by UPPAAL TIGA), that for slower controllers (e.g. replacing the guards $\mathtt{z} \geq 1$ by $\mathtt{z} \geq 2$) no strategy exists which will ensure our control purpose.

UPPAAL TIGA is integrated in the UPPAAL 4.0 framework permitting the use of discrete (shared or global) variables over simple or structured types (arrays and recods) including user-defined types. Functions can be declared using C-like syntax and used in guards and update statements. Edges have an additional select statement as a shorthand notation for all edges that satisfy the statement.

4 Modelling

In this section, we give a detailed description of the adhoc method in which the climate controller has been modelled in UPPAAL TIGA. We divide the description into a model section and a property section with guiding. Note that some of the UPPAAL TIGA code snipplets could be given a mathematical description; we have chosen the code in order to allow for the precise reconstrution of our method.

4.1 The Models

The compound model consists of three kinds of automata, the neighbor automaton, an auxilliary automaton, and controller automaton. Each of these are described in turn in the following.

Neighbor Automaton. The neighbor model is an automaton with just uncontrollable transitions that can change the observable variables of the neighboring zone. The template for the neighbor automaton is depicted in Fig. 5 and is instantiated with a parameter `id` which can take the values 0 and 1 to indicate the left and right neighbor.

Each neighbor has a variable `temp` that discretizes the temperature information of the neighbor to either `HOTTER` or `COLDER` than the control zone. Furthermore, there is a variable `n` that holds the values of the interaction variables of the neighbor. The variable `n`, which can take any of the values `WANT`, `HAVE` and `NEITHER` (encoded as the type `choice_t`), is used to indicate whether the neighbor wants air flow from the control zone, wants to deliver air flow to the control zone, or does not want to exchange air flow with the control zone.

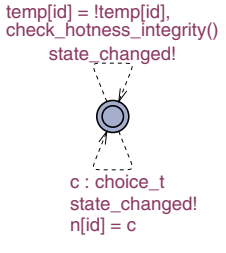


Fig. 5. Neighbor Automaton

To switch the temperature of a neighboring zone, the environment can take the uncontrollable transition at the top of Fig. 5. The call to the fucntion `check_hotness_integrity()` on the transition is explained below. The bottom

transition uses special Uppaal Tiga syntax for select statements. This is shorthand notation for the three cases where `c` takes on any of the values of `choice_t`, i.e., the environment can set the control variables of the neighbor to any kind of desired interaction. Whenever the environment changes an observable variable it synchronizes over the channel `state_changed` with the controller, to allow the controller to change the control strategy. This way we keep a strictly alternating game where the controller reacts every time an observable variable changes value.

Auxiliary Automaton. To manage the other observable variables, we introduce an auxiliary automaton that allows the environment to change these variables. The auxiliary automaton is depicted in Fig. 6.

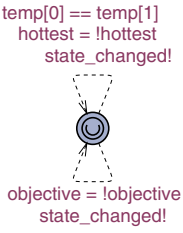


Fig. 6. Auxiliary Automaton

The final two observable variables that can change are, first, the variable `objective` which determine whether the control zone should `HEATUP` or `COOLDOWN` (bottom transition of the automaton). The second variable is a result of the discretization of the temperature information. The control zone needs information about which neighbor has hotter air. This is encoded using the Boolean observable variable `hottest` where value 0 indicates the left neighbor is hotter and vice versa for value 1. The environment can change the value of `hottest` on the top transition only when either both zones are either colder or hotter, otherwise the value can become inconsistent with the temperatures of the neighbors. The function call `check_hotness_integrity` is used by the neighbor automaton whenever the temperature changes to guarantee that `hottest` is left in a consistent state.

Controller Automaton. The controller automaton synchronizes with the auxiliary automaton and neighbor automata over the channel `state_changed` whenever an observable variable changes values. Upon synchronization, the controller enters the committed state `Decide` and the setting of the control variables is determined on the transition exiting `Decide`. The controller automaton is depicted in Fig. 7.

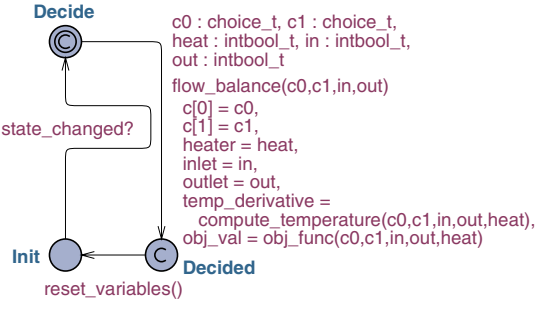


Fig. 7. Controller Automaton

The controller automaton determines the value of five control variables: Two variables for the interaction with the neighbors (`c[0]` and `c[1]`) and one variable for each of the `heater`, `inlet` and `outlet` (the latter three are all Boolean variables). The selection statement on the transition from `Decide` to `Decided` guarantees that all possible settings are considered. The guard statement

`flow_balance` guarantees that no inconsistent control state wrt. air flow is considered, i.e., whenever air is flowing out of a zone, air is flowing into the zone (see Algorithm 1) and vice versa. After updating the control variables the combined impact of the control variables and the observable variables on the temperature of the control zone is computed using the function `compute_temperature`. We refer to `obj_func` later, when we talk about guiding.

Upon entering the committed location `Decided` the transition back to `Init` is taken immediately which resets all the control variables. This is merely a step to minimize the state space since, as we shall see, the effect of the control decision is only important in `Decided`.

Algorithm 1. Procedure to guarantee that the flow balance is satisfied.

```
proc flow_balance(c0,c1,in,out) : bool
1: bool o = out || (n[0]==WANT && c0==HAVE) || (n[1]==WANT && c1==HAVE)
2: bool i =  in || (n[0]==HAVE && c0==WANT) || (n[1]==HAVE && c1==WANT)
3: return o == i
```

Discretization of the Temperature Derivative. Since the model is discretized such that the controller does not know the exact temperature of the neighboring zones, this needs to be reflected in the computed temperature derivative.

We choose to let the different control parameters contribute to the temperature derivative according to the table to the right. The values for the airflows correspond to opening the outlet fan and getting air only from the specific source. Given that the fan capacities are fixed, getting air from multiple sources will share the capacity. E.g., getting air from both (hotter) neighbors would yield a contribution of 1 from the hottest and 0.5 from the coldest, resulting in a total contribution of 1.5. Furthermore, having multiple sources of outflow increases the inflow contribution proportionally, e.g., allowing the inlet to give a total contribution of -21 by opening the outlet and providing air for both neighbors.

Heater:	5
Inlet:	-7
Hotter neighbor	
- hottest:	2
- coldest:	1
Colder neighbor	
- hottest:	1
- coldest:	2

Computing the Temperature Derivative. As we saw above, the fans have a fixed capacity that might be shared among the different sources of outflow. Since this can result in a non-integral contribution and Uppaal Tiga only handles integers, we need to multiply these contributions with an apropriate factor to guarantee integral values. Since a single source of outflow can be shared among up to three sources of inflow, we choose a constant `OUT_CONTRIBUTION=6` to denote the available contribution per outflow source as this can be integrally shared among the potential inflow sources. This has the added effect that we need to multiply the heater contribution by six as well, to keep the proportions.

The function for computing the temperature derivative is listed in Algorithm 2. Lines 1 and 2 compute the contributors to air flow in and out of the

zone. For outflow, this, in order, corresponds to 1) is the outlet open, 2) is air flowing from the control zone to the left zone, and 3) similarly for the right zone. The computation is analogous for air flowing into the control zone.

The value of `amp` computed in line 3 is the contribution for each inflow given the total outflow. Now, the return statement computes the total effect of the control decision by using the table above and the amplifier for each inflow contribution. Note that the heat contribution is also amplified to keep the the proportions defined above.

The final two negative parts of the contribution are used to indicate that giving air away cools the zone. These are used as incentives to let the controller offer air when it wants to cool. The reason is that when the controller is used in all zones we can imagine the situation when one zone needs to cool and a neighbor want the air to heat up. In the control situation when neither are interacting, one of the zones need to intiate the cooperation, and this is accomplished with the given incentives. Note that these values are negligible in the overall contribution.

Algorithm 2. Algorithm for computing the temperature derivative.

```
proc compute_temperature(c0,c1,in,out,heat) : int
1: int outflow = out + (c0==HAVE && n[0]==WANT)+(c1==HAVE && n[1]==WANT)
2: int inflow = in + (c0==WANT && n[0]==HAVE)+(c1==WANT && n[1]==HAVE)
3: int amp = (outflow * OUT_CONTRIBUTION) / inflow
4: return OUT_CONTRIBUTION*5*heat
   + amp*(c0==WANT && n[0]==HAVE ? (temp[0]? (!hottest? 2:1) : ( hottest? -2:-1)) : 0)
   + amp*(c1==WANT && n[1]==HAVE ? (temp[1]? ( hottest? 2:1) : (!hottest? -2:-1)) : 0)
   + amp*(in ? -7 : 0)
   - (c0==HAVE) - (c1==HAVE)
```

4.2 The Property

In order to synthesize the controller, we need to specify the property that the resulting controller should synthesize. An immediate choice would be[2]:

$$\phi \equiv \texttt{control : A[] Controller.Decided imply (objective ? 1 : -1)*temp\_derivative > 0} \quad (3)$$

In other words, invariantly whenever the controller enters `Decided`, the value of `temp_derivative` should be greater than zero when heating is the objective and less than zero when the objective is cooling. However, this property would be satisfied by the simple controller that never interacts with the neighbors and turn on the heater when the objective is heating and opens the inlet and outlet when the objective is cooling.

Guiding. With the property above we can determine whether we can satisfy the main objective or not. Now, we define an objective function called `obj_func` that will guide the controller synthesis process while also satisfying the property

[2] Recall that we switched the sign of the temperature derivative when the objective is to cool down.

above[3]. Given an appropriate objective function, the following property can be used to guide the controller synthesis process[4]:

$\phi \equiv$ `control : A[] ZC.Decided imply forall (c0 : choice_t) forall (c1 : choice_t)`
`forall (in : intbool_t) forall (out : intbool_t) forall (heat : intbool_t)`
`flow_balance(c0,c1,in,out) imply obj_val >= obj_func(c0,c1,in,out,heat)`

In plain words, the property states that it should hold invariantly that whenever the controller makes a decision and enters the location `Decided`, then for all other possible controller choices that satisfy the flow balance, the computed objective function is smaller or equal to the choice made. In short, the controller always chooses a configuration of the control variables that maximizes `obj_func` among all valid choices.

The simplest objective function is to use `compute_temperature`, but to compensate for the sign depending of the objective as (3) above. This guiding process will produce a controller that maximizes (minimizes) the temperature derivative for every control decision. An alternate strategy is to define the objective function over some sort of energy consumption by, e.g., penelizing turning on the heater or fan, thus, optimizing towards energy optimality.

4.3 Controlling Humidity

As mentioned in [9], the climate controller should, ideally, be extended with the ability to control the humidity in the stable as well. However, the approach outlined in [9] makes this extension a tedious strategy, since the increase in oberservable variables creates exponentially more configurations.

Changing our model to accommodate for humidity as well, requires a slight modification to the models along the lines of how the temperature was modelled. Furthermore, the objective function needs to represent the effect of temperature and humidity with a simple value. Note, that neither the controller automaton nor the property changes, as the set of controllable variables remains unchanged.

To discretize the humidity readings of the neighboring zones, analogously to the temperature representation, we introduce a Boolean `humid` variable for each zone and a `morehumid` variable to determine which of the neighbors have air with the highest humidity. Obviously, there are constraints on consistent variable assignments for the three variables in the same way as for the temperature variables.

To incorporate the variables in the model, we need an extra uncontrollable transition in the neighbor automaton, that can change the value of the respective `humid` variable. This is followed by a consistency check on the `morehumid` variable. Moreover, we add two extra uncontrollable transitions to the auxiliary automaton, one to change the `more_humid` variable, and one to change the objective

[3] To satisfy both properties we use conjunction, but do not include the conjuction to simplify properties.

[4] Note we use guiding in the sense that the function determines which of a number of valid controllers to choose, but the synthesis process itself is not guided in order to find controllers.

with respect to humidity which is encoded in the variable `decrease_humidity`. These are all the changes needed to the automata.

We incorporate humidity information in the objective function in a similar fashion the temperature model with the exception that humidity only has an upper limit, so the objective is either to decrease the humidity or ignore the humidity. Thus, when `decrease_humidity` has value false, the humidity contributes nothing to the objective function. Otherwise, the contribution is given as in Algorithm 3 where a postive value indicates a decrease in humidity. In the algorithm, `amp` is computed as for the temperature contribution. The positive contributions are from opening the inlet (contribution of 5) and getting less humid air (contribution of 2 for the least humid air and 1 for the most humid). Receiving more humid air from a neighboring zones contributes negatively. Finally, we encourage a zone to interact, if a neighboring zone has less humid air, even if the zone does not want to interact (first two parts of the sum).

Algorithm 3. Humidity contribution to the objective function

```
1: return (!humid[0] && c0 == WANT ? 1 : 0) + (!humid[1] && c1 == WANT ? 1 : 0)
   + amp*(c0==WANT && n[0]==HAVE ? (humid[0] ? (!morehumid?-2:-1):( morehumid?2:1)) :0)
   + amp*(c1==WANT && n[1]==HAVE ? (humid[1] ? ( morehumid?-2:-1):(!morehumid?2:1)) :0)
   + amp*(5*in)
```

The objective function is constructed with a weight parameter between the temperature derivative and the humidity parameter with changing the sign of the temperature as above. The weighing can be altered to generate different controllers, which later can be compared in some appropriate fashion as discussed in Section 5.

5 Results

In this section, we present some numerical results where the controller generated by Uppaal Tiga has been simulated in Simulink using realistic values for the model (2).

The Tool Chain. According to Fig. 1, to generate production code for the climate controller of the pig stable, we need to transform the output format of Uppaal Tiga to input for Simulink. Simulink allows input of so-called S-functions which are user provided C-code that can be used within the Simulink model. We have build a script which takes Uppaal Tiga strategies as input and delivers S-functions as output. The Simulink model with the S-function can be used to either run simulations of the pig stable, or generate Comedi compliant production code through Real-Time Workshop. The code generation is realized through a Comedi library for Simulink [5]. Code generation with Real-Time Workshop allows for a multitude of targets, thus, the specific target of this application in unimportant.

Numerical Results. We have synthesized two types of controllers using the models described above. One controlling only the temperature, and one controlling both temperature and humidity. In the first experiment, we synthesized a controller for temperature only as explained in Section 4. Due to limited space, we choose not to include the graphs for the experiments, as the synthesized controller is identical to that of, [9,8]. As in [9,8], the controller behaves well under simulation and keeps the zone temperatures within the given bounds.

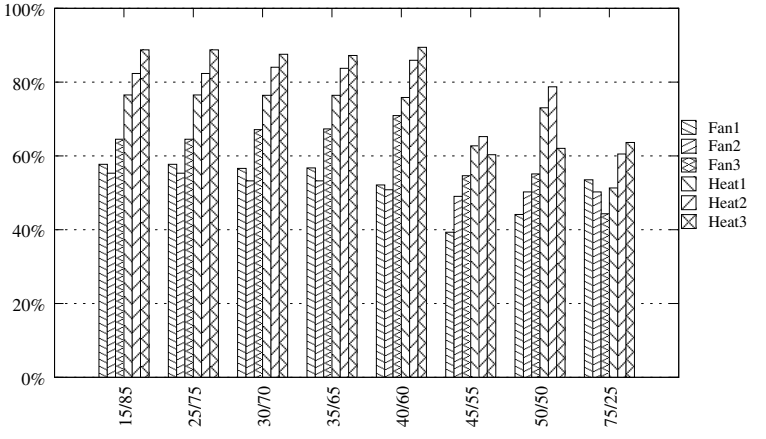
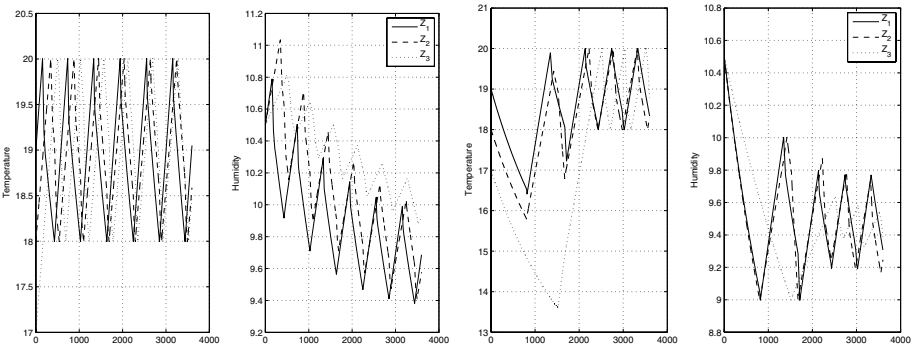


Fig. 8. Active time for heaters and fans for different controllers. The number (x/y) indicate an objective function using x percent temerature contribution and y percent humidity contribution.

In order to illustrate the guiding specification in UPPAAL TIGA a number of different controllers are simulated in Simulink. A weight is put on the objective function guiding towards temperature or humidity control. The simulation scenario is as follows: The stable is partitioned into 3 zones, and the thermal boundary is set to [18 20] and for humidity [9 10] for all three zones. The initial conditions are set to $T_1 = 19$, $T_2 = 18$ and $T_3 = 17$, $H_1 = H_2 = H_3 = 11$. All the conducted experiments stear the state to the defined boundaries in finite time, but initially some states are steared away from the boundary. In order to quantify and compare the different controllers the total time when the heat or fan are on is recorded. The result is illustrated in Fig. 8. The results show, that the controllers can be divided up into two catagories, one from 0% to 40% temperature guided, and one from 45% to 100%. The controllers in the latter category use less heat and fan capacity than the controllers in the former catagory, indicating that the former are preferred controllers. However, Fig. 9 shows how the temperature and humidity are controlled for controllers in both catagories. As it can be seen, the controller with more heat and fan activation (25/75) reaches a stable state faster than the controller with less activity of heaters and fans.[5] Thus, the

[5] Simulation results for all controllers can be found at the project website, [11].



(a) 75/25 (b) 25/75

Fig. 9. Simulation results for temperature and humidity when guiding towards a) 75% temperature and 25% humidity and b) vice versa

choice between the controllers is not immediately clear, but the quatifications can be used by the control engineers to make an informed choice.

Note that we have not tested the code in the actual pig stable. However, for the temperature controller we can rely on the results provided in [6], the the generated controller is identical to that of [9,8]. These results show that the controller does not have identical quantitative properties to the simulation, though, the qualitative properties are identical. I.e. the experiments show that the temperature oscillates as in the simulation, however, the temperature in the real stable under/overshoots the limits. This is mainly caused by the fact that the zones are not thermically isolated in the sense that air will interchange between zones, even when the zones do not what to interchange air.

For the humidity controller we do not have any experimental data to rely on, but this will be investigated in the future.

6 Conclusions and Future Work

In this paper, we have presented a complete tool chain for automatic controller synthesis from timed game automata models to production code. For the livestock production case study, the controller synthesis process has enabled, through guiding, to synthesize an identical controller do that of [9,8]. The controller in [9,8] was synthesized in a tedious manual way, which indicates the importance of a simple automated process. Note that the notion of time was not necessary in modelling our controller, however, we choose Uppaal Tiga because the tool was available and one the only ones of it's kind.

Furthermore, the model was easily extended to include humidity, which was left as a matter to explore in [9,8], but never pursued due to the heavy time requirement of the added exponential complexity. With an appropriately defined weighted objective function, Uppaal Tiga was used to synthesize a controller

capable of regulating temperature as well as humidity in a matter of seconds. A number of controllers were synthesized with variying weights between temperature and humidity, and all were able to reach stable temperature and humidity conditions in Simulink simulations. Simulink was further used to track the heat and fan activity for the different controllers, in order to allow for comparison of different controllers. This can be a very effective strategy for differentiating controllers and choosing an appropriate one among a number of controllers satisfying the conditions. As future work, we want to continue conducting experiments in the real life pig stable provided by Skov A/S in order to evaluate the different controllers capacity of controlling temperature as well as humidity in a real life setting.

References

1. Arvanitis, K.G., Soldatos, A.G., Daskalov, P.I., Pasgianos, G.D., Sigrimis, N.A.: Nonlinear robust temperature-humidity control in livestock buidings. Submitted to Biosystems Engineering (2003)
2. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, D., Lime, K.G.: Uppaal tiga: Time for playing games. In: proc. of Computer Aided Verification (CAV'07) (to appear, 2007)
3. Uppaal Tiga Homepage (2006), http://www.cs.aau.dk/~adavid/tiga
4. Janssens, K., Van Brecht, A., Zerihun Desta, T., Boonen, C., Berckmans, D.: Modeling the internal dynamics of energy and mass transfer in an imperfectly mixed ventilated airspace. Indoor Air 14, 146–153 (2004)
5. Jessen, J.J., Schiøler, H., Nielsen, J.F.D., Jensen, M.R.: Cots technologies for integrating development environment, remote monitoring and control of livestock stable climate. In: Proceedings 2006 IEEE International Conference on Systems, Man, and Cybernetics (2006)
6. Jessen, J.J.: Embedded Controller Design for Pig Stable Ventilation Systems. PhD thesis, Automation and Control, Department of Electronic Systems, Aalborg University (2007)
7. Lygeros, J., Tomlin, C., Sastry, S.: Controllers for reachability specifications for hybrid systems. Automatica 35, 349–370 (1999)
8. De Persis, C., Jessen, J.J., Izadi-Zamanabadi, R., Schioler, H.: A distributed control algorithm for internal flow management in a multi-zone climate unit. International Journal of Control. Accepted for publication (to appear)
9. De Persis, C., Jessen, J.J., Izadi-Zamanabadi, R., Schioler, H.: Internal flow management in a multi-zone climate control unit. In: Invited paper in the session Networked Control Systems 2006 CCA/CACSD/ISIC (2006)
10. SIMULINK (2007), http://www.mathworks.com/products/simulink/
11. Project Web (January 2007), http://www.cs.aau.dk/~illum/automatic_control/
12. Real-Time Workshop (2007), http://www.mathworks.com/products/rtw/

Symbolic Reachability Analysis of Lazy Linear Hybrid Automata*

Susmit Jha, Bryan A. Brady, and Sanjit A. Seshia

EECS Department, UC Berkeley
{jha,bbrady,sseshia}@eecs.berkeley.edu

Abstract. Lazy linear hybrid automata (LLHA) model the discrete time behavior of control systems containing finite-precision sensors and actuators interacting with their environment under bounded inertial delays. In this paper, we present a symbolic technique for reachability analysis of lazy linear hybrid automata. The model permits invariants and guards to be nonlinear predicates but requires flow values to be constants. Assuming finite precision, flows represented by uniform linear predicates can be reduced to those containing values from a finite set of constants. We present an abstraction hierarchy for LLHA. Our verification technique is based on bounded model checking and k-induction for reachability analysis at different levels of the abstraction hierarchy within an abstraction-refinement framework. The counterexamples obtained during BMC are used to construct refinements in each iteration. Our technique is practical and compares favorably with state-of-the-art tools, as demonstrated on examples that include the Air Traffic Alert and Collision Avoidance System (TCAS).

1 Introduction

A hybrid system is a dynamical system which exhibits both discrete and continuous behavior. Hybrid automata [4] have proved to be useful mathematical structures for modeling systems comprising discrete transition systems interacting with continuous dynamical systems. However, it is clear that in any implementation of a hybrid automaton, the state of the dynamical system reported to the discrete controller is digitized with finite precision by sensors, and the output signals of the controller transmitted to its actuators are also of finite precision. Further, the controller can only observe continuous state variables at discrete time points. Hence, it is somewhat unrealistic to assume that the controller can interact with its environment continuously and with infinite precision.

The inherent discrete nature of a controller of a hybrid system has led to recent efforts [17,2,3,1] towards studying the discrete time behavior of hybrid systems. A similar argument in favor of focusing on discrete time behavior is presented by Henzinger and Kopke [12]. *Lazy linear hybrid automata* (LLHA) [2,3] model the discrete time behavior of hybrid systems having finite precision and bounded

* Supported in part by SRC contract 1355.001, NSF grants CNS-0644436 & CNS-0627734, and Microsoft Research. The first author was also supported by the Berkeley Fellowship for Graduate Studies from UC Berkeley.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 241–256, 2007.

delays in actuation and sensing. Further, their definition of LLHA allows nonlinear invariants and guards. However, the discrete behavior in this model depends on the sampling frequency of the controller as well as the precision of variables, and hence, the discretized representations are very large and any enumerative analysis would not be feasible for systems of appreciable size.

In this paper, we present a symbolic technique for reachability analysis of *lazy linear hybrid automata*. We make the following novel contributions:

1. On the theoretical side, we present an abstraction hierarchy for LLHA that can be used for reachability analysis within a counterexample-guided abstraction-refinement framework.
2. We give an implementation of a symbolic model checker for LLHA based on bounded model checking and k-induction that operates at any level of abstraction.
3. We demonstrate the scalability of our methods in comparison to other state-of-the-art tools on examples such as Automated Highway Control System (AHS) and the Air Traffic Alert and Collision Avoidance System (TCAS).

Related Work. PHAver (Polyhedral Hybrid Automaton Verifyer) [11] is a tool for verifying safety properties of hybrid systems. It uses on-the-fly over-approximation to handle affine flows by iterative partitioning of the state space. PHAver considers a continuous time model unlike the discrete time semantics of LLHA. Our work is much more closer to the HYSDEL tool [17]. The discrete hybrid automata underlying the HYSDEL tool is formed by the connection of a finite state machine with a switched affine system through an interface. Our work is similar to HYSDEL in its considering an inertial interface between the digital and the continuous components of the hybrid system. Unlike our symbolic approach, HYSDEL uses numerical simulation for analysis. Further, our technique allows guards and invariants that use any computable function. HSolver [18,8] also allows general constraints over variables as invariants and guards. It uses interval arithmetic to check whether trajectories can move over the boundaries in a rectangular grid. Our technique uses SAT-based decision procedures for finite-precision arithmetic to do a symbolic analysis instead of an enumerative analysis. Another closely related tool is HybridSAL [21,20], which constructs discrete finite state abstractions for hybrid systems using predicate abstraction. The tool uses decision procedures and the SAL explicit state model checker. Our approach performs abstraction over the domain of variables, and uses symbolic model checking based on bit-vector decision procedures.

The examples used in this paper have been well-studied; for details on previous case studies, we refer the reader to the relevant references on TCAS [16,15] and AHS [9,14].

2 Lazy Linear Hybrid Automata

Definition 1. *A finite precision lazy linear hybrid automaton(LLHA) [3] is a tuple* $(X, V, init, flow, inv, E, jump, D, \epsilon, B, P)$. *The components of LLHA are as follows:*

- *Variables : A finite ordered set* $X = \{x_1, x_2, \ldots, x_n\}$ *of continuous variables.*
- *Control modes : A finite set* V *of control modes.*
- *Initial conditions : A labeling function init that assigns an initial condition to each control mode* $v \in V$*. The initial condition is a predicate over the variables in* X*.*
- *Flow: The possible values of rate of change of any variable in a control mode form a finite set of constant values. Let the set representing the legal flow values for variable* x_i *be denoted by* $\dot{X}_i$*. The predicate* $flow(v) \equiv (\dot{x}_1 \in \dot{X}_1) \wedge (\dot{x}_2 \in \dot{X}_2) \ldots \wedge (\dot{x}_n \in \dot{X}_n)$ *represents legal flows at location* $v \in V$*.*
- *Invariant condition : A labeling function inv that assigns an invariant condition to each control mode* $v \in V$*. The invariant condition* $inv(v)$ *is a convex predicate over the variables in* X*.*
- *Control switches : A set* E *of edges* (v, v') *from a source mode* $v \in V$ *to a target mode* $v' \in V$*. A function "*$update_{(v,v')}$*" associates a variable assignment to each control switch* (v, v')*.*
- *Jump conditions : A labeling function jump that assigns a jump condition to each control switch* $e \in E$*. A jump condition from the control mode* v *to* v'*,* $\psi_{(v,v')}$ *is a predicate over the variables in* X*.*
- *Delay parameters :* $D = \{g, \delta_g, h, \delta_h\}$ *is the set of delay parameters such that* $0 \leq g \leq g + \delta_g < h \leq h + \delta_h \leq P$*, where* h *denotes the sensing delay,* g *denotes the actuation delay and* P *is the sampling interval of the controller.*
- *Precision :* ϵ_i *is the precision of measurement of variable* x_i*.*
- *Range :* $B_i = [B_{i_{min}}, B_{i_{max}}]$ *is the range of the variable* x_i*.*
- *Period* P *represents the time period associated with the discrete controller i.e. control mode switches take place at times* $T_0, T_1, T_2, \ldots$ *where* $T_{k+1} = T_k + P$*.*

The lazy semantics of hybrid automata [2,3] means that if a control mode switch took place at time T_k, then the delay in actuating a change in $flow$ lies between $[T_k + g, T_k + g + \delta_g]$. Similarly, a control decision made at time T_{k+1} is based on the values of variables read by the controller at some time in the interval $[T_k + h, T_k + h + \delta_h]$. The parameters δ_g and δ_h represent the bounded uncertainty in actuation and sensing delay respectively. Since the sampling frequency of any implementation of a hybrid automata is always finite, this model focuses on the discrete time behavior of the hybrid automata.

The precision ϵ_i depends on the accuracy of the sensors measuring x_i from the continuous dynamical system. Guards and state invariants are evaluated on the values of the x_i variables that have been rounded using the value of ϵ_i. The parameter B reflects the range of values which can be taken by a state variable associated with a fixed width register. Unlike the conventional definition of linear hybrid automata [12], invariants and guards in LLHA can be nonlinear.

The flows in linear hybrid automata are represented using convex linear predicates over *only* the rates of change of variables (also called uniform linear predicates [13]). Under the assumption of finite precision, such flows can be considered as set of constant values of rate of change of different continuous variables. Thus, LLHA can be used for representing hybrid systems with convex linear flows. (This point is further discussed in the Appendix.) Note that the above model is same as the one formulated by Agrawal and Thiagarajan [2,3].

Definition 2. *A configuration of a hybrid automaton, with n continuous variables, is a n+1-tuple,* $c = (s, x_1, x_2, \ldots, x_n)$ *where* $s \in V$ *is the control mode,* $x_1, x_2, \ldots, x_n$ *is the valuation of the continuous variables of the hybrid automaton.*

The semantics of a hybrid automaton describes its evolution in terms of change in configuration. We use the notation $c + \alpha$ to denote the configuration in which continuous state variables are incremented by α. Also, we extend the order relation on the continuous variables to configurations. We say that $c \leq c'$ if we know that $x_i \leq x_i'$ for each x_i in c and the corresponding x_i' in c'.

We define a symbolic collection of configurations as a state of the hybrid automaton and describe the evolution of the hybrid automaton in terms of change in its state. This definition is used in Section 4 to present the bounded model checking algorithm.

Definition 3. *A state of the hybrid automaton is a pair* (v, ϕ) *consisting of a control mode* $v \in V$ *and a predicate* ϕ *over the variables* X*. We identify that the state of a hybrid automaton can change in two ways - flow or jump.*

- *flow: The changed state of a hybrid automata due to flow at control mode* v *for time* T *is* (v, ϕ^T)*, where*
  $\phi^T = \exists X_1 \exists \dot{X} \{(\phi \wedge inv(v))[X \leftarrow X_1] \wedge X = X_1 + \dot{X}T \wedge (\dot{X} \models flow(v)) \wedge inv(v)\}$.
- *jump: If* (v, ϕ) *is state of a system, and* (v, v') *is a control switch such that* $\phi \models jump(v, v')$*, then the state of the system can change to* (v', ϕ') *such that if* $update_{(v,v')}$ *was the update function over* $Y \subseteq X$*,*
  $\phi' = \exists Y_1 \{(\phi \wedge inv(v) \wedge \psi_{(v,v')})[Y \leftarrow Y_1] \wedge Y = update_{(v,v')}(Y_1)\}$,
  where $update_{(v,v')}$ *is the update function over* $Y \subseteq X$*.*

A state $s_2 = (v, \phi_2)$ is reachable from $s_1 = (u, \phi_1)$ *if and only if* there is a sequence of flow or jump transitions from s_1 to s_2.

3 Hierarchical Abstraction

We detail the theory underlying our hierarchical abstraction technique below. For brevity, proofs of some theorems have been omitted.

Agrawal and Thiagarajan [2,3] use two fundamental quantities in their analysis. The *fundamental time interval* is $\Delta =$ G.C.D of $\{P, g, \delta_g, h, \delta_h\}$. The corresponding abstraction quantum is $\Gamma =$ G.C.D of $\bigcup_i \{\epsilon_i/2, B_i^{min}, B_i^{max}, V_i^{in}, \dot{x}_i\Delta\}$.

Abstraction. We begin with basic definitions on how abstraction is performed. For ease of presentation, all variables are abstracted in the same way; the theory can be easily extended to a non-uniform abstraction.

Definition 4. Q_Π *is a surjection over the continuous variables using* abstraction quantum $\Pi = 2^k\Gamma$ *for some integer* k*. That is,*
$Q_\Pi : \mathbb{R} \to \mathbb{R}$*, and* $Q_\Pi(x_i) = k_i\Pi$ *iff* $x_i = k_i\Pi + \pi_i$*, where* $k_i \in \mathbb{Z}$ *and* $0 \leq \pi_i < \Pi$*.*

Abstract Configuration: A configuration $c^d = (s^d, x_1^d, x_2^d, \ldots, x_n^d)$ is a Π-abstraction of a concrete configuration $c = (s, x_1, x_2, \ldots, x_n)$ iff $s^d = s$ and $x_i^d = Q_\Pi(x_i)$.

Abstract Transition: Transitions are abstracted by abstracting jump and flow conditions. This must be done in order to ensure that transitions that are feasible in the concrete LLHA continue to be feasible in the abstract transition system, at the possible cost of introducing additional (spurious) behaviors.

1. The intuition behind the following definition of abstract guards and invariants is to relax the atomic constraints so that if $\Phi(x_1, x_2, \ldots, x_n)$ denotes a state invariant or guard, then the corresponding abstracted invariant or guard is $\Phi^a(x_1, x_2, \ldots, x_n)$ such that $\Phi(x_1, x_2, \ldots, x_n) \implies \Phi^a(x_1, x_2, \ldots, x_n)$.
2. The set of flow values are abstracted to overapproximate the reachable configurations. If the flow value in a set $\dot{X}$ is $\dot{x}$, it is abstracted by including flow values $\dot{x}^a$ and $\dot{x}^b$ in its place, where $\dot{x}^a \leq \dot{x} \leq \dot{x}^b$ (details given below).

We first describe how invariants and guards are abstracted, and then describe the over-approximation of flow.

Abstraction of invariants and guards. Invariants or guards can be expressed as a Boolean combination of atomic predicates in negation normal form (NNF), where each predicate is of the form $f(x_1, x_2 \ldots, x_n) \leq b$ where $b \in \mathbb{Q}$. If Φ is an invariant or guard, then $\Phi = f_{bool}(c_1, c_2, \ldots, c_n)$ where the constraint c_i is $f_i \leq b_i$ and where f_{bool} represents an NNF Boolean combination of its arguments.

Each predicate in the invariant or guard can be abstracted using the monotonicity of f with respect to each variable x_i, that is, $f_{x_i} = \frac{\delta f}{\delta x_i}$ is of the same sign over the range of interest. In particular, all polynomials which are linear in each variable, are always monotonic with respect to each variable.

In order to define abstract state invariants and guards, we first describe how to construct abstract inequalities using the above observation about invariants and guards. Without loss of generality, let us assume that $f(x_1, x_2 \ldots, x_n) \leq b$ is an inequality whose partial derivative f_{x_i} with respect to each variable x_i is of the same sign over the range of interest $[Q_\Pi(x_i), Q_\Pi(x_i + \Pi)]$. Then, its (conservative) abstraction is the *relaxed* inequality c_i' defined below:

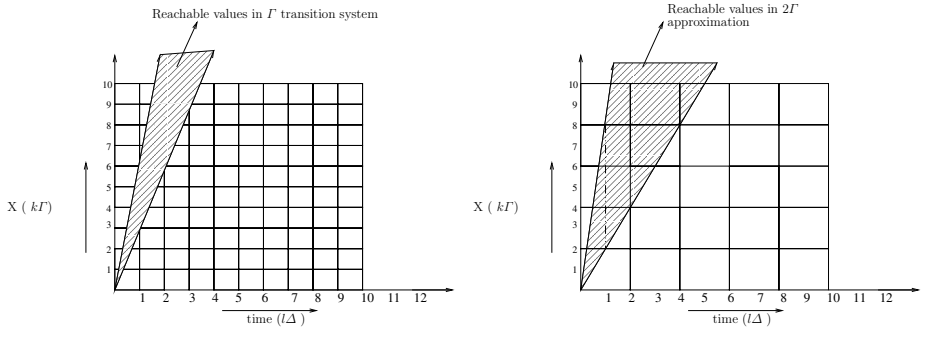
$$c_i' \equiv f(k_1, k_2 \ldots, k_n) \leq b'$$
$$\text{where } b' = Q_\Pi(b + \Pi) \qquad \text{and} \qquad \begin{aligned} k_i &= Q_\Pi(x_i) \text{ if } f_{x_i} \geq 0 \\ &= Q_\Pi(x_i + \Pi) \text{ if } f_{x_i} < 0 \end{aligned}$$

This abstraction rounds up or down each variable to the nearest multiple of Π depending on whether the function f decreases or increases with increase in the variable. The constant b is always rounded up. All assignments to the variables which satisfied the earlier constraint also satisfy the *relaxed* constraint. Hence, this is an overapproximation of the original constraint.

If $\Phi(x_1, x_2, \ldots, x_n) = f_{bool}(f_1 \leq b_1, \ldots, f_n \leq b_n)$ is the invariant or guard, the abstract state invariant or guard is defined as $\Phi^a(k_1, k_2, \ldots, k_n) = f_{bool}(c_1', c_2', \ldots, c_n')$ where the relaxed inequalities c_i' are obtained from $f_i \leq b_i$ as described above.

Thus, this relaxation results into an upper approximation of the behavior of the hybrid automaton.

Abstraction of flow conditions. If $\dot{x}$ is a rate of change allowed by $flow(s)$ for some location s, then the following two rates of change represent its abstraction $\lfloor(\frac{\dot{x}}{\Pi})\rfloor\Pi$ and $\lceil(\frac{\dot{x}}{\Pi})\rceil\Pi$. Figures 1(a) and 1(b) illustrate how flow conditions are abstracted. The abstraction of flow with 2Γ leads to an overapproximation of the dynamics of the LLHA: originally $\dot{x} \in \{3, 4, 5, 6\}$, but in the 2Γ-abstraction $\dot{x} \in \{2, 4, 6, 8\}$.



(a) 0-abstraction (b) 1-abstraction

Fig. 1. Illustration of flow abstraction

Definition 5. *A k-abstraction ($k \geq 1$) of a lazy linear hybrid automaton is an abstraction of LLHA obtained using the above explained abstraction of configurations and transitions such that $\Pi = 2^k\Gamma$. The 0-abstraction is called the Γ-transition system as the quantization is done with respect to Γ.*

We define a partial order relation $\preceq$ between transition systems below.

Definition 6. *Let TS and TS' be two transition systems such that every state of TS is mapped to some state of TS'. If every state of TS reachable from some initial state of TS has its corresponding state in TS' also reachable from an initial state of TS', then $TS \preceq TS'$.*

Prior Results. Our model of LLHA is the same as that of Agrawal and Thiagarajan [3], who initially consider a model with constant flow rate and linear invariants, and later extend the result to invariants and guards which are any "reasonable computable function". The main result of theirs which we utilize is summarized in Theorem 1.

Theorem 1. *Let a configuration of hybrid automata be $c = (s, x_1, x_2, \ldots, x_n)$ and its Γ-abstract configuration be $c^d = (s, Q_\Gamma(x_1), Q_\Gamma(x_2), \ldots, Q_\Gamma(x_n))$. A configuration c' is reachable from c iff $Q_\Gamma(c') = c'^d$ where c'^d is reachable from c^d in Γ-transition system.*

Let x_{max} and x_{min} be the maximum and minimum values that can be attained by any continuous variable and m be the number of control modes. The state space size of the Γ-transition system is $O(m^4 2^{2n}(\frac{x_{max}-x_{min}}{\Gamma})^{3n})$ [3], that is, exponential in the number of continuous variables. This huge state space makes it impractical to do any enumerative reachability analysis.

Our Results. The main result is that the *k-abstraction* of LLHA simulates the original LLHA. Further, for increasing values of k, we obtain coarser overapproximations of the LLHA which form a hierarchy of sound abstractions. Figure 2 illustrates the meaning of Theorem 2.

Theorem 2. *Let a configuration of hybrid automata be* $c = (s, x_1, x_2, \ldots, x_n)$ *and its abstraction be* $c^d = (s, Q_\Pi(x_1), Q_\Pi(x_2), \ldots, Q_\Pi(x_n))$, *where* $\Pi = 2^k\Gamma$. *If a configuration*

c' is reachable from c in time $T = l\Delta$ and $Q_\Pi(c') = c'^d$, then c'^d is reachable from c^d in the k-abstraction.

Proof. For configuration $c = (s, x_1, x_2, \ldots, x_n)$, let $\dot{x}_1, \dot{x}_2 \ldots, \dot{x}_n$ be the rates of change of continuous variables satisfying $flow(s)$ and $\widehat{\dot{x}_1}, \widehat{\dot{x}_2}, \ldots, \widehat{\dot{x}_n}$ be the rates of change of continuous variables satisfying $flow(\widehat{s})$ where $\widehat{s}$ is a predecessor state of s, that is, $(\widehat{s}, s) \in E$. Let c' be a configuration reachable from c. In case of change due to reset of variables at jump, the above theorem follows due to the adjustment to guards and invariants. We prove the above theorem for the case where the change is effected due to flow evolution.

Since, the relation $\leq$ for configurations is defined in terms of the ordering of individual variable, we consider an arbitrary variable in the rest of the proof below. If x_i is the value of the variable in c and x'_i is the value in c' after time T such that the flow rate switched after an actuation delay of t, then
$x'_i = x_i + \widehat{\dot{x}_i}t + \dot{x}_i(T - t)$
Using the definition of Γ and Δ,
$x_i = (m2^k + n)\Gamma + \gamma_i$, $\widehat{\dot{x}_i}\Delta = (2^k p' + q')\Gamma$, and $\dot{x}_i\Delta = (2^k p + q)\Gamma$,
where $0 \leq n < 2^k$, $0 \leq \gamma_i < \Gamma$, $0 \leq q' < 2^k$, $0 \leq q < 2^k$.

So, $x'_i = (m2^k + n)\Gamma + \gamma_i + (2^k p' + q')\frac{\Gamma}{\Delta}t + (2^k p + q)\frac{\Gamma}{\Delta}(l\Delta - t)$
$= (m2^k + n)\Gamma + \gamma_i + (2^k(p' - p) + (q' - q))\frac{\Gamma}{\Delta}t + (2^k p + q)l\Gamma$
Thus, $x'_i = (m + pl)2^k\Gamma + (n + ql)\Gamma + \gamma_i + (2^k(p' - p) + (q' - q))\frac{\Gamma}{\Delta}t$.

Since $0 \leq t < T$ in the above equation and $2^k\Gamma = \Pi$, x'_i lies in the interval
- $[(m + pl)\Pi, (m + (p' + 1)l + 1)\Pi)$ if $\dot{x}_i \geq \widehat{\dot{x}_i}$
- $[(m + p'l)\Pi, (m + (p + 1)l + 1)\Pi)$ if $\widehat{\dot{x}_i} \geq \dot{x}_i$

So, $Q_\Pi(x'_i)$ lies in the interval
- $[(m + pl)\Pi, (m + (p' + 1)l)\Pi]$ if $\dot{x}_i \geq \widehat{\dot{x}_i}$
- $[(m + p'l)\Pi, (m + (p + 1)l)\Pi]$ if $\widehat{\dot{x}_i} \geq \dot{x}_i$

The value of i^{th} variable in any configuration c'^d reachable from c^d in the k-abstraction, x'^d_i lies in
- $[(m + pl)\Pi, (m + (p' + 1)l)\Pi]$ if $\dot{x}_i \geq \widehat{\dot{x}_i}$
- $[(m + p'l)\Pi, (m + (p + 1)l)\Pi]$ if $\widehat{\dot{x}_i} \geq \dot{x}_i$

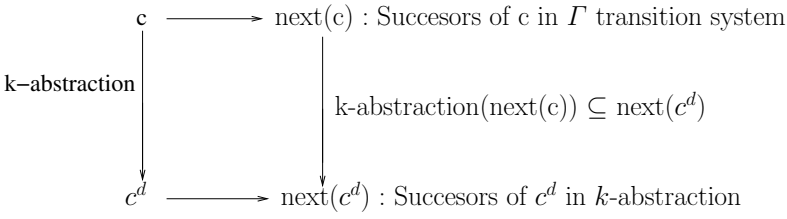


Fig. 2. Simulation by k-abstraction

Thus, for any x'_i, there exists x'^d_i such that $x'^d_i = Q_{\Pi}(x'_i)$. Using the same argument for each variable independently, the theorem immediately follows. □

Using reasoning exactly similar to the one used in Theorem 2, we can prove the hierarchy of k-abstractions presented below.

Lemma 1. *Let a configuration of k-abstraction be*
$c = (s, Q_{\Pi}(x_1), Q_{\Pi}(x_2), \ldots, Q_{\Pi}(x_n))$, *where* $\Pi = 2^k\Gamma$.
Its abstraction in $\widetilde{k}$-abstraction, where $\widetilde{k} \geq k$
$\widetilde{c} = (s, Q_{\widetilde{\Pi}}(x_1), Q_{\widetilde{\Pi}}(x_2), \ldots, Q_{\widetilde{\Pi}}(x_n))$ *where* $\widetilde{\Pi} = 2^{\widetilde{k}}\Gamma$.

If a configuration c' is reachable from c in k-abstraction, then
- $Q_{\Pi'}(c') = \widetilde{c}'$ *where* $\Pi' = 2^{\widetilde{k}-k}\Gamma$
- $\widetilde{c}'$ *is reachable from* $\widetilde{c}$ *in* $\widetilde{k}$*-abstraction.*

The Hierarchy Theorem 3 follows from Lemma 1 and Theorem 1.

Theorem 3. *k-abstraction $\preceq$ k'-abstraction if* $0 \leq k < k'$. *Thus, k-abstractions, where* $k \geq 0$, *form an hierarchical abstractions of the lazy linear hybrid automata. Further, 0-abstraction is the Γ-abstract transition system which bisimulates the original lazy linear hybrid automaton.*

Theorem 3 provides a framework for use of progressive abstraction of lazy linear hybrid automata to develop a sound and complete abstraction-refinement paradigm for reachability analysis of LLHA. Theorem 4 presents the relative reduction in state space size with k.

Theorem 4. *Let S_k be the state space size of k-abstraction and S'_k of k'-abstraction where $k' > k$. Then $log_2(S'_k/S_k) = 3n(k - k')$ where n is the number of continuous variables.*

4 Model Checking *k*-Abstractions of LLHA

Our implementation of a symbolic verifier of LLHA is based on three techniques: bounded model checking, "k-induction", and an overall counterexample-guided abstraction-refinement [7] framework. We describe each of these below.

We first present a symbolic representation of k-abstraction of the hybrid automata as stated in Definition 5. The continuous variables $X = (x_1, x_2, \ldots, x_n)$

are symbolically represented with integer variables $K = (k_1, k_2, \ldots, k_n)$ with the intended mapping being $Q_{\Pi}(x_i) = k_i\Pi$, where $\Pi = 2^k\Gamma$.

In the discussion below, we use the three components - guards (Ψ_{ij}), invariants (inv_i) and the flow conditions $flow(i)$ of the k-abstraction to define a symbolic transition relation TR. This is then used to describe the bounded model checking and inductive verification techniques.

Bounded model checking: We describe how the BMC formula is constructed, starting with a useful definition.

Definition 7. *A frame (F) is a tuple* (K, t_1, t_2, t, l) *where* $K = (k_1, k_2 \ldots k_n)$ *represent the variables;* t_1 *is the sensing delay;* t_2 *is the actuation delay* t_2*;* t *is the time before transition to next frame;* l *denotes the control mode.*

The initial state of the hybrid automata is the predicate $Init(F_0) \equiv (l = v_{start}) \wedge \phi_0(K)$, where v_{start} denotes the initial control mode and ϕ_0 the initial predicate over continuous variables.

The transition TR is defined as a predicate over the previous frame (F_{m-1}) and the present frame (F_m). It is a disjunction of all possible state switches (G_{ij}) and flow evolutions (E_i).

$$TR(F_{m-1}, F_m) \equiv \bigvee_{(i,j)\in E} G_{ij}(F_{m-1}, F_m) \vee \bigvee_{i\in V} E_i(F_{m-1}, F_m)$$

The switch predicates G_{ij} and the time evolution predicates E_i are defined in terms of three other quantities: I_i is a predicate that tests satisfiability of state invariant inv_i at control mode i, predicate g_{ij} tests satisfiability of guard ψ_{ij}, and e_{hi} deals with time evolution in control mode i with predecessor mode h.

Let us consider two functions - *compensated for sensing delay (csd)* and *compensated for actuation delay (cad).* These map a set of valuations of the continuous variables (K) to a set of possible corresponding valuations obtained after compensating for sensing and actuation delay respectively.

$$csd(K, i, t_1) = \{(k_1 - \dot{k}_1 t_1, \ldots, k_n - \dot{k}_n t_1) \mid (\dot{k}_1, \dot{k}_2, \ldots \dot{k}_n) \models flow(i)\}.$$
$$cad(K, h, i, t_2, t) = \{(k_1 + (\dot{k}_{1h} - \dot{k}_{1i})t_2 + \dot{k}_{1i}t, \ldots, k_n + (\dot{k}_{nh} - \dot{k}_{ni})t_2 + \dot{k}_{ni}t) \mid (\dot{k}_{1h}, \dot{k}_{2h}, \ldots \dot{k}_{nh}) \models flow(h) \text{ and } (\dot{k}_{1i}, \dot{k}_{2i}, \ldots \dot{k}_{ni}) \models flow(i)\}.$$

Let the current frame be $F_m = (K^m, t_1^m, t_2^m, t^m, l^m)$ and the previous frame be $F_{m-1} = (K^{m-1}, t_1^{m-1}, t_2^{m-1}, t^{m-1}, l^{m-1})$.

$$I_i(F_m) \equiv (i = l^m) \wedge \exists K'[K' \in csd(K^m, l^m, t_1^m) \wedge inv_i(K')]$$
$$e_{hi}(F_{m-1}, F_m) \equiv (i = l^{m-1} \wedge i = l^m) \wedge K^m \in cad(K^{m-1}, h, i, t_2^{m-1}, t^m)$$
$$g_{ij}(F_{m-1}, F_m) \equiv (i = l^{m-1} \wedge j = l^m) \wedge \exists K'[K' \in csd(K^{m-1}, l^{m-1}, t_1^{m-1}) \wedge \psi_{ij}(K')]$$

Note that the existential quantification over K' in the above identities simply reduces to a disjunction over possible flow values (see the description of *cad* and *csd* functions).
The switch and evolution predicates can now be defined as follows:

$$G_{ij}(F_{m-1}, F_m) \equiv I_i(F_{m-1}) \wedge I_j(F_m) \wedge g_{ij}(F_{m-1}, F_m) \wedge [K^m = update_{ij}(K^{m-1})]$$

$$E_i(F_{m-1}, F_m) \equiv I_i(F_{m-1}) \wedge I_i(F_m) \wedge [\bigvee_{h \in pred(i)} e_{hi}(F_{m-1}, F_m)]$$

where $pred(i)$ denotes the set of predecessor locations of i.

This completes the definition of the transition predicate.

Let the state to be checked for reachability be (s_r, ϕ_r). If reachability analysis is used to check safety properties, then (S_r, ϕ_r) would be the error state violating the safety property, Then, the predicate $unsafe(F) \equiv (l = s_r \wedge \phi_r(K))$ represents the error state, that is the *target* state for reachability analysis.

If d is the number of steps to which we want to check the k-abstraction for reachability of (s_r, ϕ_r), we need to check for the satisfiability of

$$BMC^d \equiv Init(F_0) \wedge \bigwedge_{n=1}^{d} (TR(F_{n-1}, F_n)) \wedge unsafe(F_d).$$

If BMC^d is satisfied, then the target state $(s_r, \phi_r(K))$ is reachable in k-abstraction and the frames $F_0, F_1, \ldots, F_d$ gives a trace from the start state to the target state.

Further, it is sufficient to do BMC for p steps to prove that a target state is not reachable where p is the diameter of the transition system. If BMC^j is unsatisfiable for all $j \leq p$, then the target state can not be reached in the transition system. Since the number of reachable states of the transition system provides an over-estimate of the diameter, it is sufficient (though unrealistic) to do BMC for number of steps equal to the state space size of the k-abstraction.

Induction: We now describe an induction procedure to guarantee the unreachability of a state in a model. This can be used to prove the satisfaction of a safety property which can be expressed as a reachability query.

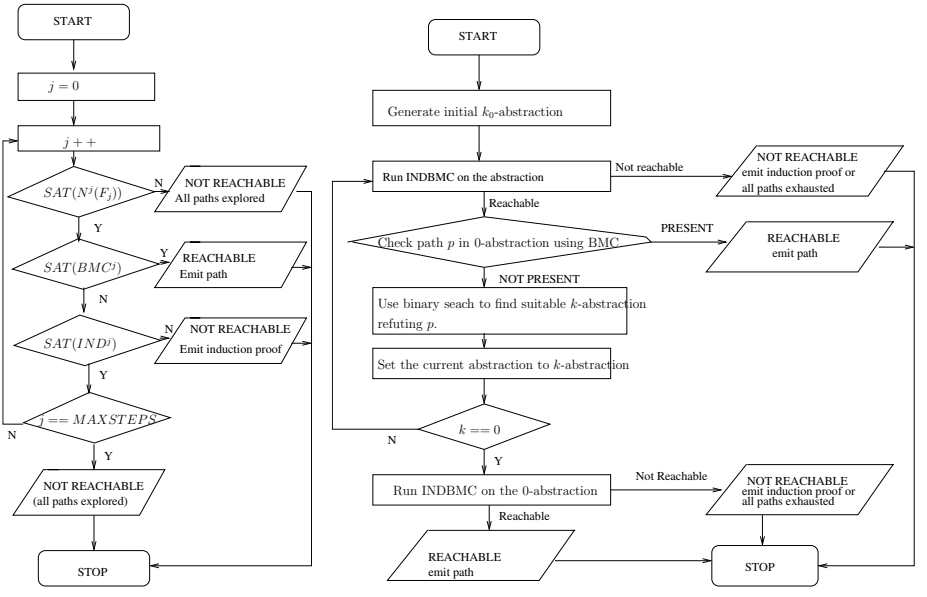
If N steps of BMC are found to be not satisfiable, that is, BMC^N is unsatisfiable, then we test the satisfiability of

$$\neg unsafe(F_0) \wedge \bigwedge_{k=1}^{N+1} (TR(F_{k-1}, F_k)) \wedge unsafe(F_{N+1}).$$

If the above is unsatisfiable, no further bounded model checking is required and all the states of the model are guaranteed to satisfy the property. Based on this, we present below a BMC algorithm along with use of induction to check for safety properties in a LLHA. We define the following predicates to be used in the algorithm.

$N^j(F_j) \equiv Init(F_0) \wedge \bigwedge_{k=1}^{j} TR(F_{k-1}, F_k)$ and $S^{j+1}(F_{j+1}) \equiv \neg unsafe(F_0) \wedge \bigwedge_{k=1}^{j+1} TR(F_{k-1}, F_k)$

If at any step of the BMC, we find that $N^j(F_j)$ is not satisfiable, it means that there does not exist a path of length j or more, and hence we can terminate with the output that the model satisfies the safety property.



(a) INDBMC module (b) Iterative refinement

Fig. 3. Symbolic reachability analysis based on BMC and induction

The bounded model checking predicate and the induction step predicate are $BMC^j \equiv N^j(F_j) \wedge unsafe(F_j)$ and $IND^j \equiv S^{j+1}(F_{j+1}) \wedge unsafe(F_{j+1})$.

The sub-routine INDBMC is presented in Figure (a). The technique is sound and complete due to the results of the preceding section; we present a detailed discussion of the abstraction-refinement framework in the next section.

Counterexample guided refinement of k-abstractions: We now describe an automated CEGAR [7] technique presented in Figure (b) which exploits the linear abstraction hierarchy presented in section 3. An initial coarse abstraction can be arbitrary chosen as k_0-abstraction depending on the size of the state space. In case the target state is not reachable in k_0-abstraction, the target state is also not reachable in the LLHA by Theorem 2. In case the target state is reachable in LLHA, then BMC will yield a path p_0 from the initial state to the target state in the k_0-abstraction. This needs to be validated with respect to the 0-abstraction. If the abstract path p_0 found in k_0-abstraction is present in 0-abstraction, then the target state is reachable in the LLHA too. If it is not present in 0-abstraction, then we select a more finer refinement k_i-abstraction which refutes the abstract spurious path. The same technique is repeated for progressively finer abstractions until the target states is shown to be unreachable or a valid path to the target sate is found. The key components of this technique are counter-example validation technique and automated refinement step. The path obtained at any iteration is a satisfying assignment to BMC^j, it can be validated on 0-abstraction by doing a BMC on BMC^j to identify the first spurious transition. If BMC^j has a

satisfying assignment for 0-abstraction too, then the path is valid. The hierarchy of abstractions allows the use of binary search to find the smallest value of l such that the l-abstraction refutes the path identified in the coarser abstraction. The complete technique for reachability analysis of LLHA based on iterative refinement and bounded model checking is presented as a flowchart in Figure **??**. The soundness of this technique is ensured by Theorem 2 and 1. Since, we start with some initial k_0-abstraction and every step involves a progress in refinement, we take at most k_0 iterations before terminating. For the iteration considering k-abstraction, $MAXSTEPS$ would be the diameter of the k-abstraction. In worst case, this algorithm needs to consider 0-abstraction which can have a very large diameter and the BMC of this transition system can be unrealistic, but the completeness of our technique is guaranteed.

Theorem 5. *The iterative abstraction refinement technique presented in Figure (b) is sound and complete.*

5 Experimental Results

In this section, we present the results of experiments on two case studies. [1] All experiments were performed on a workstation with Intel Xeon 3.06 GHz processors and 4GB RAM. UCLID bit-vector decision procedure [5] was used with MiniSat as the underlying SAT engine. Any other bit-vector decision procedure could alternatively be used as the verification engine in our technique.

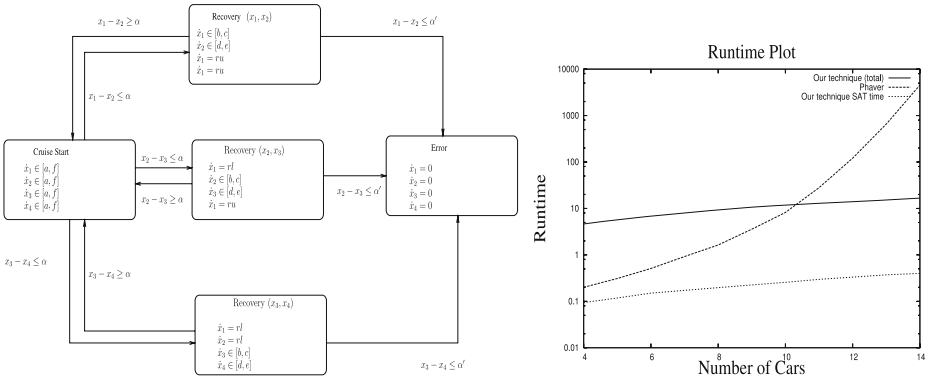
Automated Highway Control System

AHS (Figure (a)) is an arbiter which ensures that there is no collision between cars running on a highway by imposing legal speed ranges. This example has being widely used in literature [9,14]. We use the description by Jha et al [14] and extend it to handle inertial delays. The number of cars is used as a parameter to scale the example.

A set of legal parameter values is:
(All distance measures are in km, time is in hr and all speeds are in km/hr)
$\alpha = .002$, $\alpha' = .0005$, $a = 10, rl = 20, b = 30, c = 40, d = 50, e = 60, ru = 70, f = 100$
$\epsilon = 10^{-5}$, $g = 10^{-3}$, $h = 5 \times 10^{-4}$, $\delta_g = 5 \times 10^{-4}, \delta_h = 5 \times 10^{-5}$ and $P = .01$.
Correspondingly, quantization factors are $\Delta = 5 \times 10^{-5}$ and $\Gamma = 5 \times 10^{-5}$.

The safety property to be verified was that the control mode is never the "error" mode. Figure (b) compares the runtime of our technique and that of Phaver on this example for different number of cars. It shows that our approach is more scalable than Phaver. Our technique could handle large instances with 150 cars in less than 2 minutes while Phaver took more than 10 hours to analyze model with 15 cars. For this example, *we did not do any abstraction.* Γ-abstraction for AHS with even large number of cars could be easily handled by our BMC+induction technique and did not necessitate any abstraction-refinement iteration as shown by the runtime plot in Figure (b). Further, the bulk of the run-time taken by

[1] A complete set of UCLID, Phaver or HSolver modules as well as data pertaining to run-time and memory requirements can be obtained from the first author's webpage.



(a) Model (b) Runtime plot

Fig. 4. Automated Highway Control System with 4 vehicles

our technique is used up in building the model. The time taken to solve the corresponding SAT problems for BMC and Induction are a very small percentage of total run time.

Air Traffic Alert and Collision Avoidance System

TCAS is a predictive warning system used for avoiding collision of aircrafts using a sequence of preventive and corrective resolution advisories. The model for TCAS resolution used here is similar to the one used by Pappas et al [16]. We make a few changes to the model to make it more realistic. The TCAS specification [6] uses *expected time to collision* for detecting collision threats and not distance between aircrafts used in Pappas et al example [16]. The max in the constraint avoids division by zero. The k/x_r term ensures that slow approaches are avoided by triggering threat if x_r is small. This makes the problem harder since these invariants are non-linear. Hence, LHA tools like Phaver can not be used for this example.We allow the input for speed of aircrafts to be an interval. It is realistic to expect the speed of aircrafts to be in a range rather than assuming them to be a constant input. We also allow inertial delays in actuation and sensing.

The parameters used in the experiment were taken from the specifications in TCAS 2, Version 7 documentation [10] and TCAS-201 simulator [19] specifications. The time-zone considered for advisory is $30 - 120$ seconds (t_{near} and t_{far}, respectively). The distance d is taken to be 15 nautical miles (that is, 27.78 kms.). The range of speed for aircraft is allowed to range between 100 knots to 510 knots (nearly 200 km/hr to 1000 km/hr). It may be noted that the maximum speed of Airbus 380 is Mach 0.88 (nearly 505 knots). The LLHA parameters used in our example are $128\mu s \leq g, h \leq 256\mu s$ and $\epsilon = 2^{-15}$ nautical miles [19].

Phaver cannot handle this example due to non-linear invariants and guards. We modeled TCAS example using same LLHA parameter values in HSolver [8]. It did not terminate in 180 minutes with any answer. This further stresses the *hardness* of this example and underlines the significance of our technique's

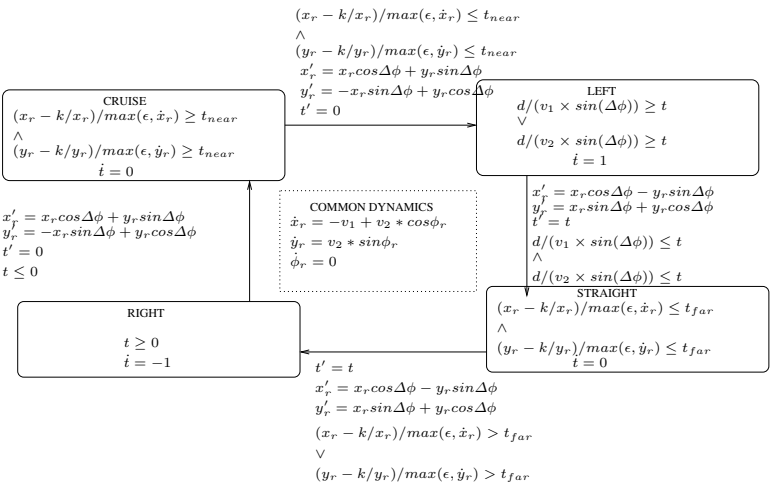
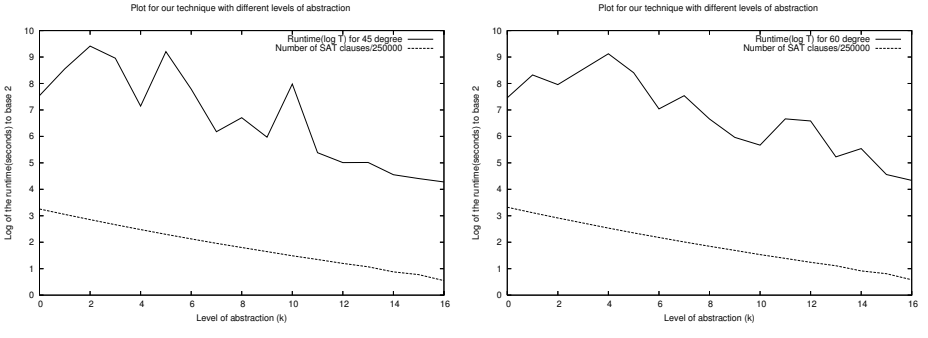


Fig. 5. Air Traffic Alert and Collision Avoidance System



(a) $\Delta\phi = 45$ degree (b) $\Delta\phi = 60$ degree

Fig. 6. Plot comparing runtimes of our technique for different levels of abstraction

scalability. Figure 6 depicts how the run-time of our tool and state space size vary for different levels of abstraction (the x-axis gives the value of k for k-abstraction). There is an initial increase due to addition of extra flows but for larger abstractions, the time taken is much less compared to the actual model. Since no refinement is needed for any value of k, the points in the graph represent run-time for only a particular abstraction level. The analysis of the 16-abstraction of the original model allows us to conclude in less than 20 seconds that the model is safe, about 10 times faster than analyzing the original model (0-abstraction).

References

1. Agrawal, M., Stephan, F., Thiagarajan, P.S., Yang, S.: Behavioural approximations for restricted linear differential hybrid automata. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 4–18. Springer, Heidelberg (2006)
2. Agrawal, M., Thiagarajan, P.S.: Lazy rectangular hybrid automata. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 1–15. Springer, Heidelberg (2004)
3. 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)
4. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.) Hybrid Systems. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1992)
5. Bryant, R.E., Kroening, D., Ouaknine, J., Seshia, S.A., Strichman, O., Brady, B.: Deciding bit-vector arithmetic with abstraction. In: Proceedings of TACAS 2007. LNCS, vol. 4424, pp. 358–372. Springer, Heidelberg (2007)
6. Chan, W., Anderson, R., Beame, P., Notkin, D.: Combining constraint solving and symbolic model checking for a class of a systems with non-linear constraints. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 316–327. Springer, Heidelberg (1997)
7. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
8. Damm, W., Pinto, G., Ratschan, S.: Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 99–113. Springer, Heidelberg (2005)
9. Deshpande, A., Godbole, D.N.A.G., Varaiya, P.: Design and evaluation tools for automated highway systems. In: Hybrid Systems, pp. 138–148 (1995)
10. Federal Aviation Administration. Introduction to TCAS II Version 7 (November 2000), http://www.arinc.com/downloads/tcas/tcas.pdf
11. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In: HSCC, pp. 258–273 (2005)
12. Henzinger, T.A., Kopke, P.W.: Discrete-time control for rectangular hybrid automata. TCS 221(1–2), 369–392 (1999)
13. Ho, P.-H.: Automatic analysis of hybrid systems. PhD thesis, Cornell Univ. (1995)
14. Jha, S.K., Krogh, B.H., Weimer, J., Clarke, E.M.: Reachability for Linear Hybrid Automata Using Iterative Relaxation Abstraction. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 287–300. Springer, Heidelberg (2007)
15. Livadas, C., Lygeros, J., Lynch, N.A.: High-level modeling and analysis of tcas. In: RTSS '99, p. 115. IEEE Computer Society Press, Washington, DC, USA (1999)
16. Pappas, G., Tomlin, C., Sastry, S.: Conflict resolution for multi-agent hybrid systems. In: CDC, pp. 1184–1189 (1996)
17. Potocnik, B., Bemporad, A., Torrisi, F., Music, G., Zupancic, B.: Hysdel Modeling and Simulation of Hybrid Dynamical Systems. In: MATHMOD Conference, Vienna, Austria (February 2003)
18. Ratschan, S., She, Z.: Constraints for continuous reachability in the verification of hybrid systems. In: Calmet, J., Ida, T., Wang, D. (eds.) AISC 2006. LNCS (LNAI), vol. 4120, pp. 196–210. Springer, Heidelberg (2006)

19. TCAS201 Specification Datasheet.
http://www.aeroflex.com/products/avionics/rf/datasheets/tcas201.pdf
20. Tiwari, A.: Approximate reachability for linear systems. In: Maler, O., Pnueli, A. (eds.) HSCC 2003. LNCS, vol. 2623, pp. 514–525. Springer, Heidelberg (2003)
21. Tiwari, A., Khanna, G.: Series of abstractions for hybrid automata. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 465–478. Springer, Heidelberg (2002)

Appendix

Constant Differential Inclusion. We provide an intuitive explanation of how constant differential inclusion of uniform linear predicates [13] are used in literature to model linear flows for reachability analysis. Such a transformation is sufficient for determining reachability as we do not reason about the time taken to reach a configuration.

This is illustrated in Figure 7. The convex polygon represents the rate of change of variables (x, y) such that is (r_x, r_y) is a permitted flow, then $(r_x t, r_y t)$ lies in the interior of the convex polygon, where t is one time unit.. The polygon is convex as we only consider convex linear flows. Also, the polygon is constant and does not change with change in configurations (x, y) as the flow condition is uniform and does not depend on (x, y). The configurations (x, y) reachable in k time units would be $(x_0 + k(r_x t), y_0 + k(r_y t))$, where (x_0, y_0) is initial configuration, $(r_x t, r_y t)$ is a point in the convex polygon. Thus, the two rays with (x_0, y_0) as origin and touching the angular extremes of the convex polygon represent the possible reachable configurations. If we form a rectangle using the vertices of the polygon that touch the rays, the set of configurations reachable for flow values lying in the rectangle is same as the previously reachable sets. The bound constraints arising from the rectangle can be, thus, used in place of the uniform linear predicate for the purpose of reachability analysis.

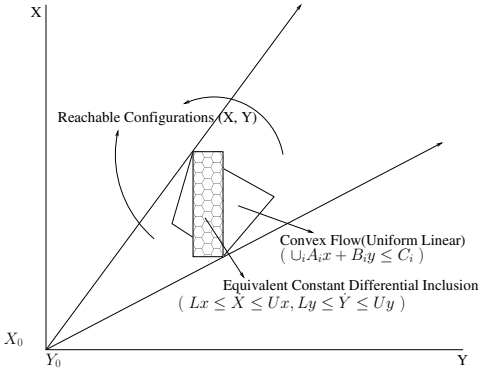


Fig. 7. The flows given as uniform convex linear predicates can be represented using constant differential inclusion for the purpose of reachability

Combining Formal Verification with Observed System Execution Behavior to Tune System Parameters*

Minyoung Kim[1], Mark-Oliver Stehr[2], Carolyn Talcott[2], Nikil Dutt[1], and Nalini Venkatasubramanian[1]

[1] University of California, Irvine, USA
{minyounk,dutt,nalini}@ics.uci.edu
[2] SRI International, USA
{stehr,clt}@csl.sri.com

Abstract. Resource limited DRE (Distributed Real-time Embedded) systems can benefit greatly from dynamic adaptation of system parameters. We propose a novel approach that employs iterative tuning using light-weight, on-the-fly formal verification with feedback for dynamic adaptation. One objective of this approach is to enable system designers to analyze designs in order to study design tradeoffs across multiple layers (for example, application, middleware, operating system) and predict the possible property violations as the system evolves dynamically over time. Specifically, an executable formal specification is developed for each layer of the distributed system under consideration. The formal specification is then analyzed using statistical model checking and statistical quantitative analysis, to determine the impact of various resource management policies for achieving desired end-to-end timing/QoS properties. Finally, integration of formal analysis with dynamic behavior from system execution will result in a feedback loop that enables model refinement and further optimization of policies and parameters. We demonstrate the applicability of this approach to the adaptive provisioning of resource-limited distributed real-time systems using a multi-mode multimedia case study.

Keywords: Iterative System Tuning, Formal Modeling, Statistical Formal Methods, System Realization, Cross-layer Timing/QoS/resource Provisioning for Distributed Systems

1 Introduction

Next generation mobile embedded applications are highly networked, and involve end-to-end interactions among multiple layers (application, middleware, network, OS, hardware architecture) in a distributed environment. Timing plays a critical role in QoS-aware system design for a large class of such distributed applications. Firstly, timing can impact application semantics. Multimedia applications have soft real-time needs often stated using parameters such as jitter,

* This work was partially supported by NSF award CNS-0615438 and CNS-0615436.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 257–273, 2007.

synchronization skews and bounded end-to-end delays. Secondly, there are several sources of unpredictability and timing violations in a distributed network; this introduces nondeterminism in timing. The ability to compensate on-the-fly for timing violations at different levels is of paramount importance. Thirdly, several system level optimizations for effective utilization of distributed resources can interfere with the timing properties of executing applications. Finally, many applications have flexible QoS needs that dictate how tolerant they are to delays and errors — the lack of stringent timing needs can be exploited for better end-to-end resource utilization.

The dual goals of ensuring adequate application QoS (expressed as timeliness, reliability, and accuracy) and optimizing resource utilization at all levels of the system presents significant challenges. In this context, our preliminary study [1] demonstrated the need for integration of formal methods with experimentally based cross-layer optimization methods [2,3]. Systematic analysis based on well-defined models ensures that corner-cases are covered and allows bounds for critical performance parameters to be determined. Recently, we proposed probabilistic formal methods to provide analysis of given cross-layered optimization policies with quantifiable confidence [4].

To leverage these prior efforts, we propose an iterative tuning approach for DRE systems that couples two important facets:

1. a light-weight, on-the-fly formal verification system that can be used dynamically to evaluate the impact of different resource management policies for achieving end-to-end timing/QoS properties, and
2. a system realization that enables feedback of additional information on dynamic system execution behaviors to enhance our light-weight formal modeling and analysis.

The integration of formal analysis combined with observed system execution behavior permits better analysis of both cross-layer and end-to-end timing/QoS properties for highly distributed systems that employ resource constrained devices.

This paper contains the following contributions:

– We present a generic framework to address iterative system tuning of distributed embedded systems by integrating two synergistic approaches: on-the-fly formal verification and learning from system realization. The light-weight formal verification provides degrees of confidence in the feasible solutions satisfying multidimensional constraints. System realization enables dynamic adaptation by refining the model of the system and the environment.
– Our work is validated and tested in the context of distributed mobile multimedia applications that have wide consumer applicability, execute in highly dynamic changing environments and present interesting opportunities for tradeoff analysis and enforcement.

The rest of this paper is organized as follows: we start by motivating our approach. Next, we present the overview of our framework, followed by a description of our case study (multi-mode multimedia communication system). In

Section 5, we explain our approach in depth. Specifically, we describe the modeling and specification of our case study. We then introduce our probabilistic formal analysis, followed by discussion of the feedback loop with our system realization. Our implementation and experimental results show the applicability of our framework to the distributed real-time multimedia communication domain. The last section summarizes our approach and discusses future research directions.

2 Supporting Adaptation Under Timing Constraints

Timing can affect, and be affected by several system and resource parameters such as storage/buffer, CPU, network topology and communication characteristics as follows.

- Power/Timing Tradeoffs: Power optimization strategies have a significant implication on the timing properties at different levels of the system. For instance, dynamic voltage scaling techniques within the operating system dictate slowdown of the CPU while lengthening the execution time, which results in possible deadline misses.
- Quality/Timing Tradeoffs: A change in the quality of communicated information represents changes in the execution time, communication time and buffering needs; for instance, lower quality video requires less decoding effort and time.
- Buffering/Timing Tradeoffs: The presence of a buffer in the datapath of the streaming information can relax the timing needs at various layers. For instance, larger buffers at the end device imply that timing constraints on receiving new data can be less stringent at the cost of memory usage. This in turn can translate into longer sleep durations (low-power operation) for power-intensive communication components. Buffering can also be used to compensate for noise, and hence delays, in the communication networks.
- Error Resilience/Timing Tradeoffs: Error resilience needs are often posed by applications to dictate fidelity requirements. In streaming applications, errors are often introduced in the communication process due to the presence of the network noise. Avoiding these errors typically involves strategies that retransmit information, encode and check for integrity of data transfer — all of which have significant implications on the overall timing behavior of the system.

This problem will be much more complex if we consider that the system and environment may keep evolving, requiring dynamic adaptation. Given a current configuration and a set of changes (e.g., new application/task; parameters for existing tasks such as framerate/resolution/synchronization; device residual power level; network delay/jitter, etc.), we need to perform dynamic adaptation (re-determine the policy, followed by bound/sensitivity analysis on the impact of the selected policy) since all the changes are critically related to timing.

In this context, we propose a unified framework for iterative and proactive system tuning to support adaptations. Initially, our framework performs property

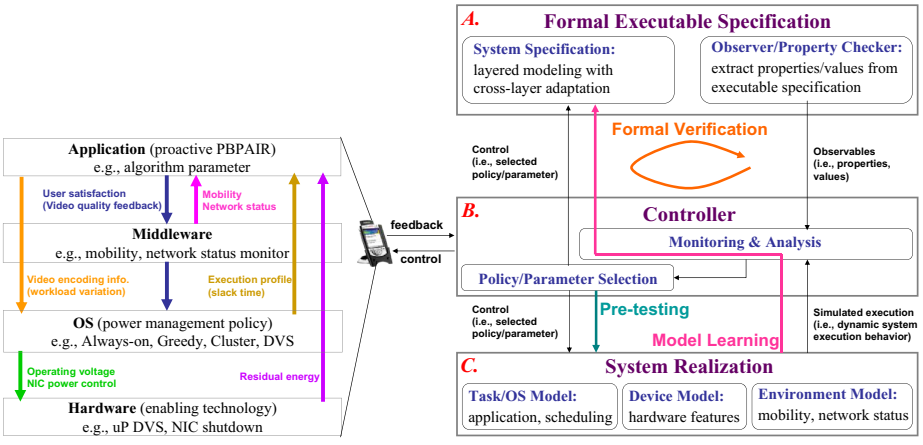


Fig. 1. The Iterative System Tuning

checking and quantitative analysis among candidate policy/parameter settings via formal executable specifications followed by probabilistic formal analysis. In our framework, the *iterative* tuning allows model refinement from up-to-date and continuous observations of system execution behavior. Furthermore, this can be used to improve adaptation by verifying given system properties or by relaxing constraints. A priori information, as forecasted by the system realization, enables *proactive* control.

3 A Framework for Iterative/Proactive System Tuning

Figure 1 presents the overall flow of our approach. We take three major steps: (1) formal modeling, (2) probabilistic formal analysis, and (3) model refinement and proactive control. In Figure 1, the *Box A* represents the formal modeling. The core of our formal modeling approach is to develop formal executable models of system components at each layer of interest. These models express functionality, timing, and other resource considerations at the appropriate level of detail and using appropriate interaction mechanisms (clock ticks, synchronous or asynchronous messages). Models of different layers are analyzed in isolation and composed to form cross-layer specifications. The use of Maude as a reasoning tool will be discussed in more detail in Section 5.1.

One advantage of formal executable models is that they can be subjected to a wide range of formal analysis, including: single execution scenarios, search for executions leading to states of interest, and model-checking to understand properties of execution paths. The *Box B* in Figure 1 shows the evaluation phase of given specifications to generate statistics of monitored properties and values. Specifically, we have developed new analysis techniques (statistical model-checking and statistical quantitative analysis) that combine statistical and formal methods, and applied them to a case study treating the videophone mode of a

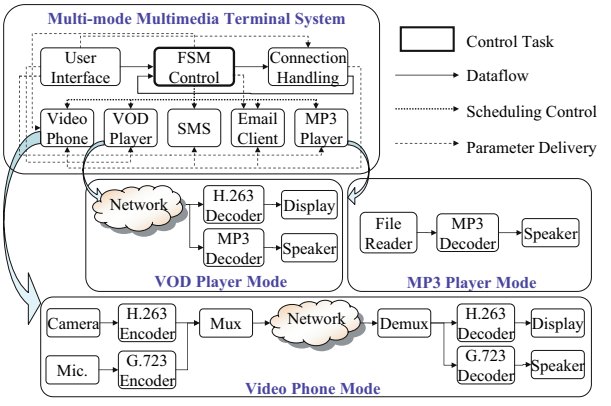


Fig. 2. Case Study: MMMT (Multi-Mode Multimedia Terminal)

multi-mode multimedia terminal [4]. Section 5.2 introduces a brief review of our probabilistic formal methods.

Using such models and analysis, tools can be developed to achieve adaptive refinement of an end-to-end system specification into appropriate policy/parameter settings. We propose an *iterative* tuning strategy that combines formal methods (verification) with dynamic system execution behavior (obtained by either simulation or implementation). The execution behavior from system realization (*Box C* in Figure 1) is fed back into the formal modeling to refine the executable specification (*model refinement*). In addition, we can assure the quality of a new policy/parameter constructed by the controller. In Figure 1, *Pre-testing* on a system realization can lead to improvements because typically the formal model can not cover all the possible implementation details of a real system (*proactive* control). We will explain iterative tuning and proactive control in Section 5.2 and 5.3 in more detail.

4 Case Study: Multi-mode Multimedia Terminal

Although we intend our approach to be widely applicable, we begin by developing and evaluating formal specification models in the context of distributed multimedia applications.

Figure 2 shows an example of a multi-mode multimedia terminal (MMMT) system [5] that we are using as a research vehicle. The figure depicts a hierarchical composition of tasks within the MMMT system. At the top level, three types of hierarchical tasks are defined to specify each mode of operation: soft real-time (a videophone, a VoD player, an MP3 player), event-driven (email client), and time-critical emergency messaging (SMS-Short Message Services). Three other tasks are also specified at the top level for user interface, connection handling, and task execution control. In addition, each mode of operation consists of multiple tasks as shown in the figure. This type of application requires frequent task set changes based on user input and/or node/network conditions (e.g., residual

power level, packet drop rate, noise level, etc.). As an example, a high-end videophone mode would be able to better meet its timing constraints at maximum CPU performance while receiving packets via a reliable channel. However, if the residual power level dropped or packet loss rate increased significantly, then we might need to save energy by reducing QoS or suspending some tasks. A user also can explicitly change modes and assign different priorities for each task/mode.

As you see from the layered view of a device in Figure 1, the resource management policies that are used in the different layers include: a specific video encoding/decoding algorithm at the application layer; network monitoring at the middleware layer; and DPM (Dynamic Power Management) and/or DVS (Dynamic Voltage Scaling)[1] at the OS layer [6]. Network traffic shaping and/or trans-coding at the middleware layer can be also utilized. Each policy has parameters that can be used to fine-tune the behavior. In addition, there are hardware parameters that can be set.

For instance, we consider *proactive* PBPAIR (Probability-Based Power-Aware Intra Refresh) [7] as an application layer policy. The *PBPAIR* scheme inserts intra-coding (i.e., coding without reference to any other frame) to enhance the robustness of the encoded bitstream at the cost of compression efficiency. Intra-coding improves error resilience, but it also contributes to reducing encoding energy consumption since it does not require motion estimation[2] (which is the most power consuming operation in a predictive video compression algorithm). The additional *proactive* feature means that we have a priori information on the user's mobility (e.g., current zone, speed and trajectory, etc.) and network situation (e.g., packet loss rate, delay, etc.) that later will be used for selection among policies and related parameter tuning before the user enters a new zone. If PBPAIR is selected as an application layer policy, then algorithm-specific parameters such as *Intra threshold* value must be chosen for appropriate execution. Note that the parameter selection at one layer affects other layers. For example, PBPAIR increases intra-coding by lowering the *Intra threshold* parameter when there is high network packet loss (monitored at middleware layer), which impacts the DVS decision at OS layer since the execution profile of the application is changed.

5 Iterative Tuning by Formal Verification Combined with System Realization

Our approach combines

– **Modeling, specification and reasoning about cross-layer and end-to-end properties:** We propose a novel approach based on concurrent

[1] DPM puts a device into a low power/performance state to save energy when the device is not serving any request during a suitably long time-period determined by the shutdown and wake-up overhead of the device. DVS aims at saving energy by scaling down the supply voltage and frequency when the system is not fully loaded.

[2] In predictive coding, motion estimation eliminates the temporal redundancy due to high correlation between consecutive frames by examining the movement of objects in an image sequence to try to obtain vectors representing the estimated motion.

rewriting logic to formally specify and reason about end-to-end timing/QoS issues across layers and study their inter-relationships.
- **Design of policies and mechanisms for addressing tradeoffs based on the cross-layer analysis:** Our work examines the impact of various resource management techniques on end-to-end timing/QoS properties and enables informed selection of resource management policies along with rules for instantiation of parameters that derive the policies.
- **Model refinement and proactive control:** We enhance our light-weight formal modeling and analysis by integrating it with observations of system execution behavior to achieve adaptive reasoning and proactive control by providing more precise information on current execution and future state.

In the following subsections, we explain each component; formal executable specification (Section 5.1), controller (Section 5.2), and system realization (Section 5.3) of our proposed framework (Figure 1) in depth beginning with our modeling effort.

5.1 Modeling Effort

Our formal modeling approach utilizes Maude [8] to formally specify the environmental changes as well as the policies/parameter settings that can be made at each of these levels in isolation and for the combined layers. Maude is a specification language based on rewriting logic with supporting analysis tools. The Maude system has been used in the specification and analysis of a wide range of logics, languages, architectures and distributed systems [9,10].

Rewriting logic [11] is a simple logic well-suited for distributed system specification. The state space of a distributed system is formally specified as an algebraic data type by giving a set of sorts (types), operations, and equations. The dynamics of such a distributed system is then specified by rewrite rules of the form

$$t \longrightarrow t' \quad \textbf{if } c$$

where t, t' are terms (patterns) that describe the local, concurrent transitions possible in the system, and c is a condition constraining the application of the rule. Specifically, when a part of the distributed state matches the pattern t, and satisfies c, then this part can change to a new local state t'. Rewriting logic specifications are executable, as proofs in rewriting logic are carried out by applying rewrite rules which can also be viewed as steps of a computation.

The Maude system is based on a very efficient rewriting engine, supporting use of executable models as prototypes. It also provides the capability to search the state space reachable from some initial state by the application of rewrite rules. This can be used to find reachable states satisfying a user-defined property. The system also includes an efficient model-checker for checking properties expressed in linear temporal logic. The Maude system, its documentation, and related papers and applications are available from the Maude website `http://maude.cs.uiuc.edu`.

```
*** Property checker
  op batteryExpires : Configuration → Bool .
  eq batteryExpires(< CPU : HW | residualEnergy : F, atts > C:Configuration)
     = (if (F ≤ 0.0) then true else false fi) .

*** Observer
  msg Obs : Bool → Msg .
  msg EnergyConsumption : Float → Msg .
  msg BatteryExpires : Bool → Msg .

  rl [cpuObs] :
    < CPU : HW | consumedEnergy : F, policy : P, atts >
    ⇒
    EnergyConsumption(F)
    BatteryExpires(batteryExpires(< CPU : HW | consumedEnergy : F, atts >)) .
```

Fig. 3. Maude Specification: Property Checker and Observer

In the object-oriented specification style supported by Maude, the system state (configuration) is typically represented as a multiset of objects and messages. Passage of time is modeled by functions that update the configuration appropriately, for example decrementing timers or decreasing remaining power. Rules can either be instantaneous or tick rules of the form

$$C \rightarrow delta(C,T)\ in\ time\ T \quad \textbf{if}\ T \leq mte(C)$$

where C is a term representing the system configuration. This tick rule advances time non-deterministically, according to a chosen time sampling strategy, by a time T less than or equal to *mte(C)*, the maximal time allowed to elapse in one step, in configuration C, and alters the system state, C, using the function *delta*[3]. Both *delta* and *mte* are user-defined to capture how time passes in a particular model.

In Maude syntax, objects have the general form

$$< ObjectName : ClassName \mid Attribute_1 : Value_1, ..., Attribute_n : Value_n >$$

where *ObjectName* is an object identifier, *ClassName* is a class identifier, and each *Attribute : Value* pair specifies attribute identifier and its value.

At the end of each execution, we examine the final configuration of a Maude specification that has several objects and messages. From those objects and messages, we need to extract meaningful data — observables. Observables can be properties or values. For example, to check whether the battery expires or not at the end of the execution, we need to check the *residualEnergy* attribute in *CPU* object at hardware layer. If the value for the *residualEnergy* attribute is positive, then the battery is not empty. Otherwise, the *batteryExpires* property returns *true* meaning the system used up the battery. We encode the check of properties into the model so that the result contains *true* or *false* depending on whether a property holds or not. On the other hand, if we want to have the energy consumption rather than the answer for property hold, we can utilize the observer

[3] The idea of a tick rule is taken from Real-Time Maude [12].

such as the one shown in Figure 3. The observer replaces each object with suitable messages that have data values for the observables. Furthermore, we use the Maude API, a foreign language interface to embed the Maude rewriting engine into larger applications, to extract observables from the Maude execution and to generate statistics of results.

5.2 Adaptation by Statistical Evaluation and Reinforcement

Once we extract observables from runs of a formal executable specification, the controller performs formal verification and pre-testing as illustrated in Figure 1. For verification purposes, we use *probabilistic formal methods*. The controller also interacts with a system realization to pre-test selected policies/parameters, and to obtain information on dynamic system behavior to improve the formal model. These two techniques are summarized below.

Probabilistic Formal Methods. To evaluate feasible design points, we adapt and improve two statistical evaluation methods — statistical model checking and statistical quantitative analysis [4]. For statistical model checking, probabilistic properties such as "*Probability* that a system can survive with given residual energy in t time units is more than θ %" are examined. Those formulae are essentially a restricted version of Continuous Stochastic Logic (CSL) [13] without nesting. Indeed we found no need for nested formulae or an exact numerical solution for our application domain. Therefore, we use statistical model checking to verify such probabilistic properties, more precisely hypothesis testing based on Monte-Carlo simulation results.

For statistical quantitative analysis, we estimate the expected value of certain observables such as "*Average* energy consumption in t time units within confidence interval (δ) and error bound (α)". Statistical evaluation can be performed with a large quantity of data that follows a normal distribution, and hence allows the estimation of the expected value and our confidence. To determine the mathematical soundness of the approximation, we perform a Jarque-Bera (JB) normality test [14]. More importantly, we generate traces on demand to reduce the evaluation time since it is linearly proportional to the trace generation time (i.e., Monte-Carlo simulation time with a different seed). Detailed explanation on statistical theory background and our implementation can be found in [4].

Model Refinement. Within our framework, there are at least two roles for feedback from observation of system execution behavior: it can be used to improve the model (to make it more accurately match the real environment), and it can also be used to directly improve the policy using optimization or learning methods. In this particular work, we only consider the former case, that is *model refinement* using the information from dynamic system behavior.

For instance, the formal specification initially models the execution times of the tasks as a normal (Gaussian) distribution with the average of $\frac{(BCET+WCET)}{2}$ and the boundary value of $3\times\delta$, where δ represents the standard deviation, based on profiled best case execution time (BCET) and worst case execution time

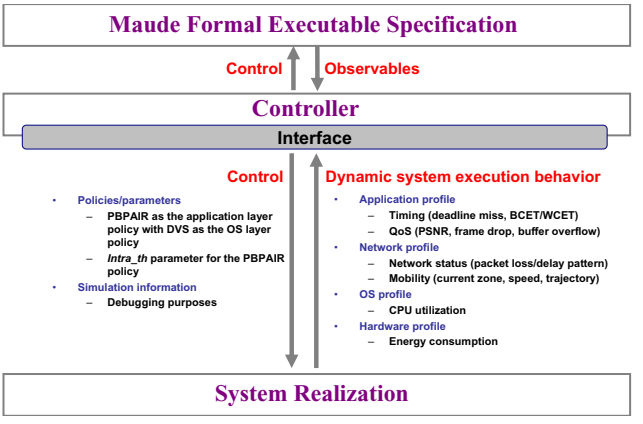


Fig. 4. Interactions between Controller and System Realization

(WCET) from sample runs. This model is refined in turn by replacing BCET and WCET with observations from dynamic system execution (either by system realization in 5.3 or real implementation), in order to more realistically reflect the actual executions characterizing the system in practice such as data dependent execution times. In our framework, the controller (written in Java) uses a Java/Maude foreign language interface to execute/update Maude specifications and to extract the results for analysis.

5.3 System Realization

As we briefly mentioned in Section 3, the integration of formal analysis with a system realization (as illustrated in Figure 1) will result in a feedback loop that includes the formal models, simulation, and monitoring of running systems for analysis of the system behavior and for optimizing the choice of policies and parameters. Specifically, the system realization takes policies/parameters and returns the dynamic system execution behavior at each layer as seen in Figure 4. For instance, if the controller selects PBPAIR (with appropriate *Intra_Th* parameter) as the application layer policy and DVS as the OS layer policy, the system realization executes using the appropriate settings and reports profiled information such as consumed energy, timing/QoS aspects.

For this purpose, we define a collection of library routines and their arguments that can be used to implement a system realization [15]. At the application layer, we need to create a task set for a chosen mode. As an example, video phone mode has four tasks; video encoder/decoder and audio encoder/decoder, each with its own parameters (e.g., PBPAIR has *Intra_Th* parameter.). Besides, the input/output data structure is task specific. For instance, an H.263 encoder with PBPAIR policy takes the *Intra_Th* parameter, the network packet loss rate (precisely, this information will be provided as middleware layer input), and raw video sequences as inputs to generate a bitstream robustly encoded against network transmission errors. In addition, there are encoder QoS related parameters

(e.g., quantization value, IP ratio, frame-rate, buffer size). As a result, application profile data such as QoS (PSNR(Peak Signal to Noise Ratio), frame drops) and timing (deadline misses, BCET, WCET) aspects should be reported.

The most two important observations from our system realization are timing (BCET, WCET) and network related information. The framework uses timing information to refine the model and network related information to generate *proactive* control. Therefore, in the experiments, we will demonstrate how the framework can achieve iterative system tuning for proactive control using those two pieces of feedback from system execution behavior.

6 Experimental Results

6.1 Evaluation Platform

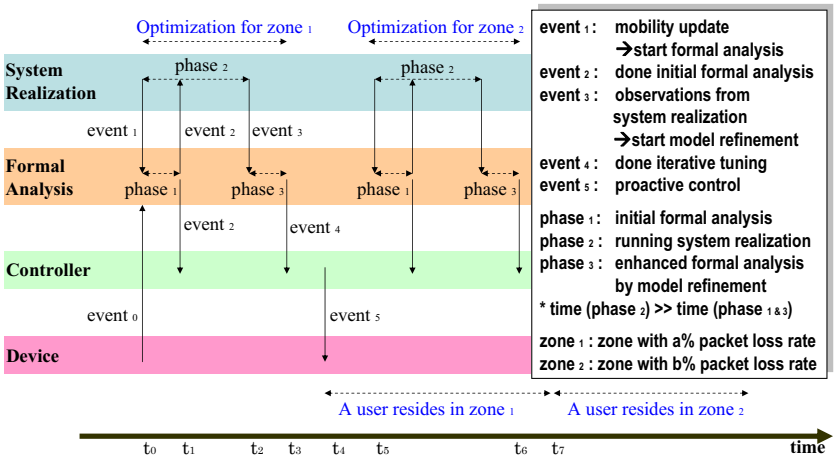
Using formal executable specifications in Maude, we model *PBPAIR* as an application layer policy as well as two power management schemes — *Greedy* and *Cluster* — as OS layer policies. In the *Greedy* scheme, the power manager shuts down whenever the device is idle, while the *Cluster* scheme tries to aggregate idle periods to maximize energy efficiency. A subset of the MMMT system — video encoder and decoder for videophone mode — is modeled with the workload variation of a PBPAIR encoder [7] and an H.263 decoder [16]. The network zone information is assumed to be given and the hardware implementation is from [17,18].

For the system realization, we use the Simics [19] full system simulation platform, capable of simulating target systems that include real network connection and run operating systems and workloads. Specifically, we use the Simics model of a PowerPC-based Ebony card [17] with a PPC440GP processor [18] that boots Linux 2.4. The execution profile from the Simics environment is reported and used to the formal specification via a real network (port forwarding feature in Simics). As explained in Section 5.2, execution profile from the system realization is fed back into the formal model to enhance the solution quality.

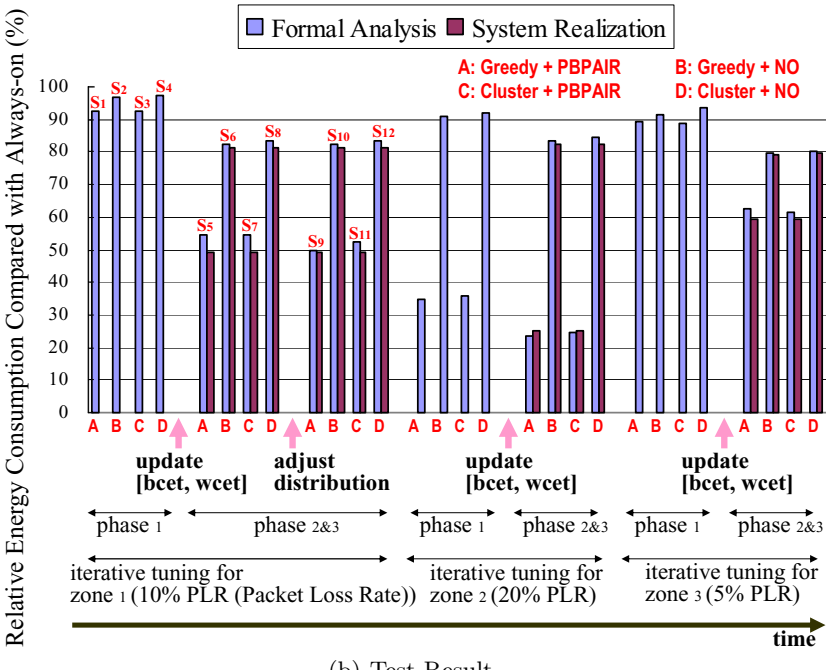
6.2 Model Refinement

Modeling with formal executable specifications, rather than implementing simulators of distributed systems under consideration, enables us to carry out formal analysis (e.g., statistical model checking and quantitative analysis). However, there exist opportunities to improve the formal model to adapt to the system dynamics. For this purpose, we allow model refinement from observed system execution behavior by equipping the controller with a loop to experiment with the system realization.

Figure 5 illustrates model refinement based on the dynamic system execution behavior from a system realization. The formal specification initially models the execution times of each task as a function of BCET and WCET from samples, and performs verification/evaluation of the given policies based on that model shown as $phase_1$ in the Figure 5(a). The Maude traces followed by statistical



(a) Test Scenario



(b) Test Result

Fig. 5. Experimental Results: Model Refinement and Proactive Control

quantitative analysis provide the initial estimations up to time t_1 in Figure 5(a). Since obtaining execution behavior from the system realization usually takes much longer time than formal analysis (e.g., in our case, it is of the order of hundreds times slower than formal analysis), it is beneficial to find the best policy by formal analysis first. At time t_2, the system realization starts generation of the BCET and WCET that reflect the actual executions as described as $event_3$

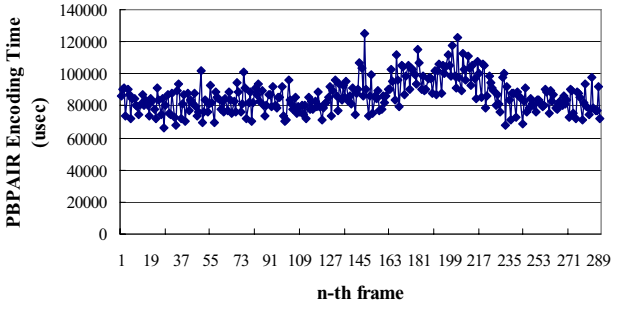


Fig. 6. Dynamic Execution Behavior of PBPAIR

in Figure 5(a). Then, the formal model is refined by updating BCET and WCET to enhance the analysis results as shown between t_2 and t_3 ($phase_3$).

Let us take an example. At time t_0, the formal specification models PBPAIR execution with [BCET, WCET] as [109 msec, 202 msec][4], and provides the analysis results s_1 to s_4 for four different policy/parameter selections (A, B, C, D) in Figure 5(b), respectively. Since our system realization reports dynamic execution of PBPAIR as shown in Figure 6, we can refine [BCET, WCET] to be [66 msec, 125 msec] that leads to formal analysis results s_5 to s_8 in Figure 5(b). Furthermore, we adjust the parameter of the frame encoding time distribution model in the formal specification since many frame encoding times are close to BCET as shown in Figure 6. Instead of simply providing simulated [BCET, WCET] as the parameter of the normal distribution model explained in Section 5.2, we use the *actual* average execution time observed from the system realization. Formal analysis results s_9 to s_{12} in Figure 5(b) show a better approximation (i.e., closer to the estimation based on dynamic execution behavior from system realization) due to this adjustment.

It should be pointed out that $phase_1$ (initial formal analysis) and $phase_3$ (enhanced formal analysis) takes much less time than $phase_2$ (running a system realization). The goodness of a policy/parameter selection, however, remains same through $phase_1$ to $phase_3$. This indicates that on-the-fly, light-weight formal verification can be effectively used in adaptation by rapidly narrowing down the search space of potential policies and parameters. Furthermore, the quality of adaptation can be improved by combining formal analysis with observed system execution behavior.

6.3 Proactive Control

As we mentioned in Section 4, we exploit a priori information on a user's mobility (e.g., current zone, speed, and trajectory, etc.) and network situation (e.g., packet loss rate, delay, etc.) to select among policies and related parameter tuning before the user enters a new zone. The mobility information is used to identify the

[4] These profiled values are from [7] for various video inputs.

network situation in the current zone and to anticipate the next zone based on a user's speed and trajectory. Ideally, we need prediction techniques like time series analysis [20] to model the future trends in network traffic with some defined level of confidence ($event_0$ in Figure 5(a)). This is, however, beyond the scope of this paper. Currently, we assume that the next zone information is forecast by the system realization ($event_1$).

Figure 5 also illustrates proactive control initiated by network status update. At time t_0, the middleware layer is informed about the next zone information that a user will reach, $zone_1$ with 10% packet loss rate at time t_4. Our framework performs the iterative tuning process — formal execution followed by statistical analysis ($phase_1$) with subsequent model refining ($phase_2$ and $phase_3$) — for the next zone. As a result, our framework can generate controls ($event_5$) to the device before the user enters the new zone (any time between t_3 and t_4). Similarly, at time t_5 the formal model is informed that a user will be in a zone with 20% packet loss rate at time t_7. By the time t_6, our framework can provide *proactive* controls for $zone_2$.

7 Related Work

The authors of [21] explore probabilistic model checking (PMC) in solving the DPM problem. They obtain the optimal DPM policy by formulating the optimization problem as a discrete time Markov chain (DTMC) model and solve it using an equation solver (e.g., MAPLE [22]). Once a policy has been constructed, its performance is validated using a probabilistic model checking (PMC) tool PRISM [23]. Even though PMC enables experimenting with the effectiveness of a selected DPM algorithm in a quantitative way, the challenge still remains to determine how to actually implement a good power manager that considers complex system dynamics since their work is essentially a validation process for a specific policy using an equation solver. Besides, their analysis of stochastic systems is carried out using numerical solution techniques that are far more memory intensive. On the other hand, our approach is to start with an executable formal model specifying a space of possible behaviors and analyze these possible behaviors using light-weight statistical techniques.

Model-predictive control approaches [24,25] also attempt to address power management issues. In [24], the authors propose predictive learning to shutdown a device by exhaustively searching over a limited look-ahead horizon. In [25], a closed loop feedback control based on queuing theory is presented to optimize CPU frequency. Their solution quality depends on the future events forecasted by a mathematical model (e.g., filter). The applicability of these approaches, however, is limited to systems having a small number of control inputs with synthetic workloads.

The authors of [26] propose an incremental methodology to analyze the effect of a simple time-out based DPM scheme. They start with the functional model without timing and perform a noninterference check for behavior. Then, they extend it to a Markovian model (i.e., the execution time of each action is

modeled as an exponential function) and the effect of DPM is evaluated by standard numerical techniques. Lastly, they extend it to a general model by using profiled information from real-world measurements and simulate it to compare the result with that of Markovian model. Our framework can be seen as a generalization of their work. First, since we use Maude formal executable specifications that can have any distribution in timing by controlling the *tick* rule, our formal model corresponds with their general model (a Markovian model can be treated as a specific distribution in the general model). Their mathematical soundness is only guaranteed when the model follows exponential timing. More importantly, our primary focus is on-line adaptation based on abstract formal models complemented by a system realization, not the validation at design time.

8 Conclusions and Future Work

This paper presents a unified framework to develop formal analytical methods for understanding cross-layer and end-to-end timing issues in highly distributed systems that incorporate resource limited devices, and to integrate these methods into the design and adaptation processes for such systems. We propose iterative/proactive system tuning for DRE systems and apply them in a case study treating the videophone mode of a multi-mode multimedia terminal. The integration of formal analysis with the observation of system execution behavior results in a feedback loop that includes the formal models, simulation, and monitoring of running systems for analysis of system behavior and optimizing choice of policies and parameters. The underlying formal executable models are moderately simple to develop, and the analyses seem feasible. The experiments on a fairly complex case study demonstrate the capability of our framework — formal verification combining with observation from system realization — to dynamic tuning of DRE systems.

Ongoing and future work in this project includes:

- considering of a richer class of timed properties
- policy improvement via learning
- modeling and analysis of cross-cutting concerns (e.g., reliability, security)
- carrying out a large scale demonstration with heterogeneous applications (mission critical, multimedia) on multiple devices in a distributed network.

References

1. Kim, M., Dutt, N., Venkatasubramanian, N.: Policy construction and validation for energy minimization in cross layered systems: A formal method approach. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '06) Work-in-Progress Session
2. Forge Project: `http://forge.ics.uci.edu`
3. Mohapatra, S., Cornea, R., Oh, H., Lee, K., Kim, M., Dutt, N.D., Gupta, R., Nicolau, A., Shukla, S.K., Venkatasubramanian, N.: A cross-layer approach for power-performance optimization in distributed mobile systems. In: IEEE 19th International Parallel and Distributed Processing Symposium (IPDPS'05) (2005)

4. Kim, M., Stehr, M.O., Talcott, C., Dutt, N., Venkatasubramanian, N.: A probabilistic formal analysis approach to cross layer optimization in distributed embedded systems. In: 9th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'07). LNCS, vol. 4468, pp. 285–300 (2007)
5. Kim, D., Kim, M., Ha, S.: A Case Study of System Level Specification and Software Synthesis of Multimode Multimedia Terminal. In: IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTImedia'03), pp. 57–64 (2003)
6. Kim, M., Ha, S.: Hybrid run-time power management technique for real-time embedded system with voltage scalable processor. In: ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'01), pp. 11–19 (2001)
7. Kim, M., Oh, H., Dutt, N., Nicolau, A., Venkatasubramanian, N.: PBPAIR: an energy-efficient error-resilient encoding using probability based power aware intra refresh. ACM SIGMOBILE Mob. Comput. Commun. Rev. 10(3), 58–69 (2006)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: The maude 2.0 system. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 76–87. Springer, Heidelberg (2003)
9. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: specification and programming in rewriting logic. Theoretical Computer Science 285(2), 187–243 (2002)
10. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All about maude, a high-performance logical framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
11. Meseguer, J.: Conditional Rewriting Logic as a unified model of concurrency. Theoretical Computer Science 96(1), 73–155 (1992)
12. Real-Time Maude: http://www.ifi.uio.no/RealTimeMaude
13. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Verifying continuous time markov chains. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 269–276. Springer, Heidelberg (1996)
14. Jarque, C., Bera, A.: A test for normality of observations and regression residuals. Internat. Statist. Rev. 55(2), 163–172 (1987)
15. Kim, M., Stehr, M.O., Talcott, C., Lee, K., Dutt, N., Venkatasubramanian, N.: Iterative system tuning for proactive systems by formal verification and system prototype: System prototype program interface. CECS Technical Report, UC Irvine (February 2007)
16. Signal Process. Multimedia Lab. Univ. British Columbia: TMN 10 (H.263+) encoder/decoder, version 3.2.0 (September 1998)
17. Ebony Board: http://www.amcc.com/Embedded/
18. http://www.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_440_Embedded_Core
19. Simics Full System Simulation Platform: http://www.simics.net
20. Han, Q., Venkatasubramanian, N.: AutoSeC: An Integrated Middleware Framework for Dynamic Service Brokering. IEEE Distributed Systems On-line 2(7) (2001)
21. Norman, G., Parker, D., Kwiatkowska, M., Shukla, S., Gupta, R.: Using probabilistic model checking for dynamic power management. Formal Aspects of Computing 17(2), 160–176 (2005)
22. Paleologo, G.A., Benini, L., Bogliolo, A., Micheli, G.D.: Policy optimization for dynamic power management. In: 35th Annual Conference on Design Automation (DAC'98). pp. 182–187 (1998)

23. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006 and ETAPS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
24. Abdelwahed, S., Kandasamy, N., Neema, S.: Online control for self-management in computing systems. In: 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04), p. 368 (2004)
25. Lu, Z., Hein, J., Humphrey, M., Stan, M., Lach, J., Skadron, K.: Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'02). pp. 156–163 (2002)
26. Acquaviva, A., Aldini, A., Bernardo, M., Bogliolo, A., Bonta, E., Lattanzi, E.: Assessing the impact of dynamic power management on the functionality and the performance of battery-powered appliances. In: International Conference on Dependable Systems and Networks (DSN'04). p. 731 (2004)

Multi-processor Schedulability Analysis of Preemptive Real-Time Tasks with Variable Execution Times

Pavel Krcal[1], Martin Stigge[2], and Wang Yi[1]

[1] Uppsala University, Sweden
{pavelk,yi}@it.uu.se
[2] Humboldt University Berlin, Germany
mstigge@informatik.hu-berlin.de

Abstract. In this paper, we study schedulability analysis problems for multi-processor real-time systems. Assume a set of real-time tasks whose execution times and deadlines are known. We use timed automata to describe the non-deterministic arrival times of tasks. The schedulability problem is to check whether the released task instances can be executed within their given deadlines on a multi-processor platform where each processor has a task queue to buffer task instances scheduled to run on the processor. On the positive side, we show that the problem is decidable for systems with non-preemptive schedulers or tasks with fixed execution times. A surprising negative result is that for multi-processor systems with variable task execution times and a preemptive scheduler, the schedulability analysis problem is undecidable, which is still an open problem in the single-processor setting.

1 Introduction

Real-time systems are often designed as a collection of real-time tasks and a scheduling strategy implemented as a scheduler. Each of the tasks may have a set of task parameters such as release rate (or release pattern), best and worst-case execution times on the target hardware, priority, deadline, etc. In the operation of a system, the tasks will be released according to the pre-designed release rates, and the released task instances are scheduled to execute on a processor by the scheduler. Here the scheduler, namely the scheduling strategy, is the critical component for the correct functioning of a system. It makes the decision about in which order the released task instances should be executed, based on the task parameters and the current system state.

An important design problem is to analyze whether all the task instances can be executed within the given deadlines, which is essentially to estimate the worst-case response times of the tasks. This is the so-called schedulability analysis based on (1) the task release patterns, (2) the task parameters and (3) the scheduling policy, all of which are from the system design phase. The source of complexity in solving the analysis problem is in dealing with task releases

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 274–289, 2007.

and preemptions. A newly released task instance may preempt a running task, which influences the response time for the preempted task. Classic solutions, e.g., Rate-Monotonic Analysis often assume deterministic task release patterns such as periodic tasks [LL73] or deterministic patterns with fixed type of non-determinism such as jitters [But97] and offsets [RC01]. A challenge is to solve the schedulability analysis problem for systems with dynamic and non-deterministic task releases and preemptions. In recent years, in a series of work, we have used timed automata [AD94] to model task release patterns and solved the problem for a large class of single-processor systems [FKPY07, FMPY06].

In [FKPY07], timed automata are extended with asynchronous tasks. Each location of a timed automaton may be associated with a task. As soon as the automaton visits a location, an instance of the associated task is released and put (scheduled) into a task queue for execution. Compared with classic task models studied in the literature on scheduling, extended timed automata provide a much more expressive model which, in fact, inherits the full expressive power of timed automata for modeling of dynamic and non-deterministic task releases and preemptions. In our previous work, the classic notion of schedulability analysis has been extended to the automata model, and it is shown that for a large class of systems, the problem can be solved automatically using algorithmic methods. However, the study has been restricted to the single-processor setting.

In this paper, we study the schedulability analysis problem of the extended automata model in the multi-processor setting. Basic scheduling algorithms for multi-processor systems can be found in, e.g., [BB06]. For recent work on schedulability analysis for multi-processor systems and further references, see [ABJ01, BCL05]. We assume that a system has a fixed number of processors available. Each processor is associated a task queue, where the released tasks wait to be processed. A scheduling policy (modeled as a function) decides for a newly released task instance into which queue and at which position in the queue it will be inserted. However, task migration is not allowed, that is, once a task instance is assigned to a processor, it will remain in the associated queue until it finishes.

On the positive side, we show that the problem is decidable as for the single-processor case if

1. the scheduler runs a non-preemptive scheduling strategy, or
2. the tasks have fixed execution times, that is, the best and worst-case execution times coincide.

It is an open problem, whether for the class of systems with variable task execution times and a preemptive scheduler, the schedulability problem is decidable in the single-processor setting. This problem becomes undecidable when the scheduling policy has at least two processors with their task queues available. More precisely, as a main technical result of this paper, we show that the schedulability problem is undecidable for multi-processor systems if

1. the scheduler runs a preemptive scheduling strategy, and
2. the tasks have variable execution times ranging over an interval between the best and worst-case execution times.

2 Preliminaries

In this section, we introduce the concept of *task automata* (timed automata extended with tasks) developed in [FKPY07], and the multi-processor schedulability problem.

2.1 Task Automata

Let $\mathcal{C}$ be a finite set of clocks. A function $\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$ is called a *clock valuation* while the set of all clock valuations over the clocks $\mathcal{C}$ is denoted by $\mathcal{V}(\mathcal{C})$. With $\nu[r]$ we denote the clock valuation which is equal to ν, except that all clocks in $r \subseteq \mathcal{C}$ are reset to zero. $\mathcal{B}(\mathcal{C})$ is the set of *clock guards* g, which are conjunctions of expressions of the form $x_1 \bowtie N$ and $x_1 - x_2 \bowtie N$ with $x_1, x_2 \in \mathcal{C}$, $N \in \mathbb{N}_{\geq 0}$ and $\bowtie \ \in \{<, \leq, >, \geq\}$. If g contains only $x_1 \bowtie N$ expressions, we say it is *diagonal-free*. A valuation ν satisfies a guard g (written $\nu \models g$) if for all expressions $x_1 \bowtie N$ and $x_1 - x_2 \bowtie N$ in g it holds that $\nu(x_1) \bowtie N$ and $\nu(x_1) - \nu(x_2) \bowtie N$, respectively.

Tasks and task queues. We define a task type as a tuple (P, B, W, D) written $P(B, W, D)$, where P is the task name (unique for each task type), $B, W \in \mathbb{N}_{\geq 0}$ the best and worst case computation times (with $B \leq W$ and $W \geq 1$), and $D \in \mathbb{N}_{\geq 1}$ the relative deadline. Note that D is a relative deadline meaning that whenever an instance of P is released, it should be computed within D time units. A task instance $P_i(b_i, w_i, d_i)$ of type $P_i(B_i, W_i, D_i)$ is a released copy of this task type with $b_i, w_i, d_i \in \mathbb{R}$ being the remaining computation times and deadline. A task queue q is a list $[P_1(b_1, w_1, d_1), \ldots, P_n(b_n, w_n, d_n)]$ of task instances (of possibly the same type). By a discrete part of a queue $[P_1(b_1, w_1, d_1), \ldots, P_n(b_n, w_n, d_n)]$ we mean the list of the corresponding task names $[P_1, \ldots, P_n]$ (the information about the remaining computation times and deadline is projected out). Let $\mathcal{P}$ be the finite set of task types and $Q_{\mathcal{P}}$ be the set of queues over this task type set.

To talk about computation and resource consumption, we shall use a function $\mathsf{Run} : Q_{\mathcal{P}} \times \mathbb{R}_{\geq 0} \mapsto Q_{\mathcal{P}}$ which given a real number t and a task queue q returns the task queue after t time units of execution on a processor. The result of $\mathsf{Run}(q, t)$ for $t \leq w_1$ and $q = [P(b_1, w_1, d_1), Q(b_2, w_2, d_2), \ldots, R(b_n, w_n, d_n)]$ is defined as $q' = [P(b_1 - t, w_1 - t, d_1 - t), Q(b_2, w_2, d_2 - t), \ldots, R(b_n, w_n, d_n - t)]$. For example, let $q = [Q(2, 3, 5), P(4, 7, 10)]$. Then $\mathsf{Run}(q, 3) = [Q(-1, 0, 2), P(4, 7, 7)]$ in which the first task has been executed for 3 time units. For an empty queue, denoted by $[]$, $\mathsf{Run}([], t) = []$.

Definition 1. *A* task automaton *over actions $\mathcal{A}ct$, clocks $\mathcal{C}$, and task types $\mathcal{P}$ is a tuple $\langle N, l_0, E, I, M, x_{done} \rangle$ where*

- *N is a finite set of locations,*
- *$l_0 \in N$ is the initial location,*
- *$E \subseteq N \times \mathcal{B}(\mathcal{C}) \times \mathcal{A}ct \times 2^{\mathcal{C}} \times N$ is the set of edges,*
- *$I : N \to \mathcal{B}(\mathcal{C})$ is a function assigning a clock constraint to each location which is called* location invariant*,*

– $M : N \hookrightarrow \mathcal{P}$ *is a partial function assigning locations with task types,*[1] *and*
– $x_{done} \in \mathcal{C}$ *is the clock which is reset whenever a task finishes.*

We write $l \xrightarrow{g\,a\,r} l'$ *for* $\langle l, g, a, r, l' \rangle \in E$.

Semantics. An important part of the operational semantics is the scheduling function, which decides into which queue and at which position a newly released task should be inserted. For the sake of presentation, we first introduce the scheduling function $\mathsf{Sch} : \mathcal{P} \times Q_{\mathcal{P}} \to Q_{\mathcal{P}}$ for the single-processor case. Given a task instance and a task queue, it returns the task queue with the task instance inserted and the order of the other task instances preserved. Depending on whether the scheduler is preemptive or non-preemptive, the function may insert new tasks as the first element or not. Since we want this function to be encodable in timed automata, the definition has the following important condition.

Definition 2. $\mathsf{Sch} : \mathcal{P} \times Q_{\mathcal{P}} \to Q_{\mathcal{P}}$ *is a* scheduling function*, if for each task type* $P(B, W, D)$ *and discrete part* $[P_1, \ldots, P_n]$ *of a queue there can be effectively constructed a diagonal-free timed automaton with*

– *Clocks* $y_1^b, y_1^w, y_1^d, \ldots, y_n^b, y_n^w, y_n^d$,
– $n + 2$ *locations* $l_0, l_1, \ldots, l_{n+1}$*, and*
– $n + 1$ *edges from* l_0 *to* l_i *for* $1 \leq i \leq n + 1$,

such that $\mathsf{Sch}(P(B, W, D), [P_1(b_1, w_1, d_1), \ldots, P_n(b_n, w_n, d_n)])$ *inserts* $P(B, W, D)$ *into the queue at the* m*-th position if and only if* l_m *is the only location reachable from* (l_0, ν) *where* $\nu(y_i^b) = b_i, \nu(y_i^w) = w_i, \nu(y_i^d) = d_i$ *for all* $1 \leq i \leq n$.

This definition generalizes the notion of a scheduling policy. Most of the standard scheduling policies such as EDF (Earliest Deadline First), FPS (Fixed Priority Scheduling), FIFO, etc. satisfy this condition, but also ad hoc policies such as combinations of the standard ones are included. Therefore, the results hold for a very general class of scheduling policies.

In the multi-processor case, the scheduling function takes k queues and a task type as an input and returns the queues with the new task instance inserted at some position in one of the queues. It can use the information from all the queues for its decision. Each timed automaton corresponding to a scheduling policy then contains clocks for all the instances in all the queues. To simplify the notation, we assume that for k queues $q_1, \ldots, q_k$ with discrete parts $\tilde{q}_1, \ldots, \tilde{q}_k$, all the task instances are indexed by one index i ranging from 1 to $\sum_{1 \leq j \leq k} |\tilde{q}_j|$, where $|\tilde{q}_j|$ denotes the number of the task instances in the queue q_j. E.g., the index of the first task instance in q_2 is $|\tilde{q}_1| + 1$. Also, all possible positions where a new task can be inserted are indexed by one index ranging from 1 to $\sum_{1 \leq j \leq k} |\tilde{q}_j| + k$. E.g., the $|\tilde{q}_1| + 2$-th position denotes the first (head) position of the second queue (the first task instance in q_2 after insertion). By b_i, w_i and d_i we denote as before the continuous queue information for the task instance p_i.

[1] Note that M is a partial function meaning that some of the locations may have no tasks.

Definition 3. *(Multi-processor Scheduler) Let $k \in \mathbb{N}$ be the* number of processors. *Then $\mathsf{Sch}_k : \mathcal{P} \times (Q_{\mathcal{P}})^k \to (Q_{\mathcal{P}})^k$ is a* multi-processor scheduling function, *if for each task type $P(B, W, D)$ and* discrete parts *of queues $\tilde{q}_1, \ldots, \tilde{q}_k$, there can be effectively constructed a diagonal-free timed automaton with*

- *Clocks $y_1^b, y_1^w, y_1^d, \ldots, y_K^b, y_K^w, y_K^d$ where $K = \sum_{1 \le j \le k} |\tilde{q}_j|$,*
- *$K + k + 1$ locations $l_0, l_1, \ldots, l_{K+k}$ and*
- *$K + k$ edges from l_0 to l_j for $1 \le j \le K + k$,*

such that the function $\mathsf{Sch}(P(B, W, D), q_1, \ldots, q_k)$ inserts $P(B, W, D)$ at the m-th position if and only if l_m is the only location reachable from (l_0, ν) where $\nu(y_i^b) = b_i, \nu(y_i^w) = w_i, \nu(y_i^d) = d_i$ for all $1 \le i \le K + k$.

Note that this definition does not allow for moving tasks between processors after they got assigned to one processor at the moment when they are released, that is, we do not allow task migration. This restriction is necessary for our decidability proofs.

A task automaton may – just as a timed automaton – perform event and delay transitions, and additionally *task finishing* transitions. An event transition corresponds to the arrival of a new task, and a delay transition corresponds to active tasks being executed while the others are waiting, or just processor idling in case some queue is empty. Transitions of the third type remove finished tasks from the queues and reset the clock x_{done}, giving thus a feedback to the automaton (x_{done} can be checked in the guards). We give now the formal definition of the operational semantics for a system with k processors as a labeled transition system (LTS), where $S := N \times \mathcal{V}(\mathcal{C}) \times (Q_{\mathcal{P}})^k$ is the state space, thus incorporating the queues into the state information. Let ν_0 be a clock valuation assigning all clocks the value 0, and $\Sigma := \mathcal{A}ct \cup \mathbb{R}_{\ge 0} \cup \{\mathsf{fin}\}$ a set of labels (for events, time pass values and task finishing).

Definition 4. *Given a scheduling strategy Sch_k over k processors, the* semantics of a task automaton $A = \langle N, l_0, E, I, M, x_{done} \rangle$ *is a labeled transition system $[\![A_{\mathsf{Sch}_k}]\!] = \langle S, s_0, \Sigma, T \rangle$ with $s_0 = (l_0, \nu_0, [])$ and T the set of transitions defined by the following rules:*

- $(l, \nu, q_1, \ldots, q_k) \xrightarrow{a}_{\mathsf{Sch}_k} (l', \nu[r], \mathsf{Sch}_k(M(l'), q_1, \ldots, q_k))$ *if* $l \xrightarrow{g\,a\,r} l'$, $\nu \models g$, *and* $\nu[r] \models I(l')$,
- $(l, \nu, q_1, \ldots, q_k) \xrightarrow{t}_{\mathsf{Sch}_k} (l, \nu + t, \mathsf{Run}(q_1, t), \ldots, \mathsf{Run}(q_k, t))$ *if* $t \in \mathbb{R}_{\ge 0}$, $(\nu + t) \models I(l)$, *and for all i with $q_i = P(b, w, d) :: q_i'$ it holds that $t \le w$, and*
- $(l, \nu, q_1, \ldots, P(b, w, d) :: q_i, \ldots, q_k) \xrightarrow{\mathsf{fin}}_{\mathsf{Sch}_k} (l, \nu[x_{done}], q_1, \ldots, q_i, \ldots, q_k)$ *if* $b \le 0 \le w$ *and* $\nu[x_{done}] \models I(l)$,

where $P(b, w, d) :: q$ denotes the queue with the task instance $P(b, w, d)$ on the first position and q being the (possibly empty) tail.

Finally, we can also define the main question this work deals with, namely whether a task automaton models schedulable sequences of task releases. As

all deadlines in our model are hard, we say that a task automaton is schedulable for a given scheduling strategy if no matter how does the (non-deterministic) computation evolve, all deadlines are met. We use q_{err} to denote queues containing a task instance $P(b, w, d)$ with $d < 0$.

Definition 5. *(Schedulability) A task automaton A with initial state $(l_0, \nu_0, [], \ldots, [])$ is* unschedulable *with* Sch_k *if* $(l_0, \nu_0, [], \ldots, []) \longrightarrow^*_{\mathsf{Sch}_k} (l, \nu, q_1, \ldots, q_{err}, \ldots, q_k)$ *for some l and ν. Otherwise, we say that A is* schedulable *with* Sch_k.

We call a queue *unschedulable* if it will inevitably lead to a deadline miss provided that all tasks take their worst case computation times. Otherwise, a queue is said to be *schedulable*. The important observation for schedulable queues is that their length is bounded:

Lemma 1 ([FKPY07]). *Given a finite task type set $\mathcal{P}$, one can effectively construct a natural number $B_{\mathcal{P}}$ such that $|q| \leq B_{\mathcal{P}}$ for all schedulable queues q.*

2.2 Decidability for Task Automata

A system, modeled by a task automaton and a scheduling function, can have the following three properties:

Preemption: The scheduler may insert a newly released task to the head of a non-empty queue, thus preempting the currently running task (a non-preemptive scheduler may insert tasks only at other positions).

Variable task execution times: There can be task types $P_i(B_i, W_i, D_i)$ such that $B_i < W_i$, meaning that the task may non-deterministically finish execution at any time within the interval $[B_i, W_i]$.

Feedback: The precise finishing time of a task may influence the new task releases (by means of using x_{done} in guards and invariants).

Note that the more of these properties are dropped for a task automaton, the easier the schedulability analysis problem becomes.

In [FKPY07], the schedulability problem is studied for single-processor systems. It is shown that (even with only one processor) schedulability becomes undecidable if a task automaton has all three properties. In turn, it has also been shown that if there is no preemption, the problem becomes decidable. The same holds if there is no variable execution time. The open question remains, whether schedulability is decidable if x_{done} is not used in the guards and invariants for creating feedback. This last variant has also been proven decidable for certain types of schedulers in [FKPY07], but there is no result for the general case.

We study these questions for multi-processor scheduling. Since single-processor systems are just a special case of multi-processor systems, the negative result, namely that the schedulability becomes undecidable if a task automaton has all three properties, transfers to our setting. We show that the problem remains decidable for the following classes:

1. A non-preemptive scheduler (but possibly variable execution time and feedback) or
2. Fixed execution time tasks ($B = W$ for all task types, but possibly a preemptive scheduler and feedback).

On the other hand, we show that the schedulability analysis problem becomes undecidable for the third case:

3. No feedback (but possibly a preemptive scheduler and variable execution time).

3 Decidable Cases of Multi-processor Scheduling

In this section, we show that the multi-processor schedulability problem is decidable for the first two cases. The proof is done by reduction of this problem to the reachability problem for timed automata. For a given number of processors, a task automaton, and a multi-processor scheduling policy, we construct a timed automaton with an error location such that the system is unschedulable if and only if the error location is reachable. Our construction is based on the construction for the single-processor case from [FKPY07] and [EWY99].

3.1 Fixed Computation Time

First, we handle the case where all tasks have fixed computation time (but the scheduler can be preemptive and there can be feedback from the scheduler back to the automaton).

Theorem 1. *The problem of checking whether a task automaton A with fixed computation times of tasks together with a multi-processor scheduler Sch_k is schedulable, is decidable.*

Proof (Sketch). The given task automaton A is transformed into a standard timed automaton $E(A)$ by taking the underlaying timed automaton of A, removing the labels and adding new labels $\mathsf{release}_P$ on edges where A would release an instance of task type P. Because of Definition 2 and Lemma 1, the whole scheduling strategy for schedulable queues can be encoded as a timed automaton $E(\mathsf{Sch}_k)$ (the *scheduler automaton*) as follows. The continuous part of the queues, namely the best/worst case computation times and remaining relative deadlines, are encoded in clocks. There are two clocks x_i^c and x_i^d for each task instance p_i in the queues, which measure how long this task instance has been computed and how long it has been released. The discrete parts of the queues (the order of the task instances) are encoded in locations of $E(\mathsf{Sch}_k)$. Since the queue length of schedulable queues is bounded (Lemma 1) and the number of the queues is fixed, there are only finitely many locations needed. Once a queue becomes unschedulable, $E(\mathsf{Sch}_k)$ will enter a dedicated error location. The edges and their guards correspond to the comparisons which the scheduling function uses for its decisions.

The values of b_i, w_i and d_i for each task instance p_i in the queue can be expressed using the clocks x_i^c and x_i^d, which is ensured by the construction in the following way. Whenever a new instance p_i of type P_j is released (event $\mathsf{release}_{P_j}$), the clock x_i^d is reset. Before a task instance p_i is put to the head of its queue (which models the beginning of the task execution), the values b_i, w_i and d_i from the queue can be expressed as B_j, W_j and $D_j - x_i^d$, respectively. The clock x_i^c is not used at all.

As soon as the task instance p_i is scheduled for execution for the first time, the clock x_i^c is reset, keeping track of its computation progress. For a running task instance p_i, the values b_i, w_i and d_i from its queue can be expressed as $B_j - x_i^c$, $W_j - x_i^c$ and $D_j - x_i^d$, respectively. Therefore, this task instance may finish whenever "$B_j \leq x_i^c \leq W_j$" holds. To model task finishing, $E(\mathsf{Sch}_k)$ removes p_i from its queue and resets x_{done}. Whenever a constraint "$x_i^d > D_j$" is met for a released task instance p_i, an error location "unschedulable" is entered.

Because of the preemption, tasks waiting in the queues may have already been executed for some time, but they are stopped now. Since it is not possible to stop their x_i^c clocks, the time for which a preempted task p_i was already computed is represented not just by its x_i^c clock, but by a difference $x_i^c - x_j^c$. Here, p_j is the task which directly preempted p_i.

When a running task p_m finishes, all x_i^c of the preempted tasks p_i are updated by subtracting x_m^c, i.e., the computation time which was needed by p_m. In general, the reachability for timed automata with such updates is undecidable, but because of the fixed computation time property, $x_m^c = B_j = W_j$ is a constant natural number (assume that the type of p_m is P_j). Therefore, clocks in the resulting timed automaton $E(\mathsf{Sch}_k)$ (and in the product automaton $E(\mathsf{Sch}_k) \parallel E(A)$) are updated only by subtractions of integers. There is also a bound on the values of the clocks which are subtracted (the deadline), which makes the reachability problem for this type of automata (*Timed Automaton with Bounded Subtraction*) decidable, as proven in [FKPY07].

The product automaton $E(\mathsf{Sch}_k) \parallel E(A)$ described above is a timed automaton with bounded subtraction for which it holds that the "unschedulable" error state is reachable if and only if A is unschedulable with scheduling strategy Sch_k.

The proof of this fact for the multi-processor case is the same as for the single-processor case in [FKPY07], with the difference that there can be several tasks running at the same timepoint. Therefore, it is necessary to check that Condition C_3 from the proof in [FKPY07], i.e., correctness of the invariant $b_i = B_m - (x_i^c - x_j^c)$, also holds here.

But from the fact that the scheduling function cannot move tasks from one queue to another one (here we need the assumption that the task migration is not allowed) it follows that whenever a task instance p_i of type P_m is preempted by another instance p_j of type P_n, it will stay preempted (i.e., not being computed) until p_j finishes. Moreover, for p_j it holds that $b_j = B_n = W_n$ at the timepoint when it preempts p_i. This together implies the correctness of C_3 and thus also of the updates at the end of preemption. □

Note that here, the diagonal-freeness in guards of the scheduler (see Definition 2) is important, because it is necessary to use clock differences to express the computation time of tasks. (A comparison of a clock difference to a constant is already a diagonal constraint which cannot be further extended by another clock.)

3.2 Non-preemptive Scheduler

In this case, tasks are not allowed to preempt other already running tasks, which makes handling the computation time of all tasks in the queues easier. Therefore, even variable computation times of tasks can be allowed.

Theorem 2. *The problem of checking whether a task automaton together with a non-preemptive multi-processor scheduler is schedulable, is decidable.*

Proof (Sketch). Here, the same construction as above is used, i.e., two automata $E(A)$ and $E(\mathsf{Sch}_k)$ are created. Again, $E(\mathsf{Sch}_k)$ keeps track of computation progress and time pass since the release in clocks x_i^c and x_i^d for each released task p_i.

As in the previous case, whenever a new instance p_i of type P_j is released (event $\mathsf{release}_{P_j}$), the clock x_i^d is reset. Before a task instance p_i is put to the head of a queue, the values b_i, w_i and d_i from the queue can be expressed as B_j, W_j and $D_j - x_i^d$, respectively. The clock x_i^c is not used at all.

As soon as the task instance p_i is scheduled for execution, the clock x_i^c is reset, keeping track of the computation progress of the task. For a running task instance p_i, the values b_i, w_i and d_i from its queue can be expressed as $B_i - x_i^c$, $W_i - x_i^c$ and $D_i - x_i^d$, respectively.

Because the scheduler is non-preemptive and the scheduling function cannot move tasks from one queue to another one, only the running task in each queue has already started its computation. The tasks which are not at the head of some queue did not start their computations yet. For this reason, we do not need to use x_i^c of the waiting tasks to compute b_i and w_i. □

Note that for this result, the definition of the scheduling function (Definition 2) does not have to be that restrictive in the sense that also diagonal constraints can be allowed for the decisions in the scheduling function. The reason is that the computation time and deadline values in the queues are encoded into (at most) one clock each. The possibility of using diagonal constraints in the scheduler's decisions makes (for the non-preemptive case) encoding even of the Least Slack First [But97] scheduling policy possible.

4 Undecidability

In this section we show, that the schedulability problem is undecidable for multi-processor systems with at least two scheduling queues. This holds even if the precise task finishing times cannot influence releases of new tasks.

Theorem 3. *The problem of checking whether a task automaton without feedback using a k-multi-processor scheduler is schedulable, is undecidable for* $k \geq 2$.

To develop the proof, we first sketch a construction used in [FKPY07] for the single-processor case, where the automaton was allowed to use task feedback. We then extend the result to the multi-processor case, even if no task feedback is involved.

4.1 Undecidability on Single-Processor Using Feedback

The undecidability proof for the general single-processor schedulability problem is done by a reduction from the halting problem for two-counter machines. A *two-counter machine* consists of a finite state control unit and two unbounded non-negative integer counters. The three possible instructions are counter-increment, counter-decrement and conditional branching (checking whether a counter value is zero). After each step, the machine state is changed deterministically. One of the states is a dedicated *halt state*. It is known, that the problem whether this halt state is reachable (the *halting problem for two-counter machines*) is undecidable.

The idea is, given a two-counter machine M, to construct a task automaton A_M and a scheduling policy such that a dedicated halt location in A_M is reachable if and only if the halt state of M is reachable. It will be ensured that no task can miss its deadline as long as the halt location is not visited. In turn, the queue can unboundedly grow in the halt location, making tasks miss their deadlines. The result of these two properties is that A_M will be unschedulable if and only if M can reach its halt state.

For each state of M's control unit there is one corresponding location l_i in A_M. These locations are connected depending on the operation which M would execute at the corresponding state (increment, decrement, branching), through auxiliary locations "executing" this instruction.

To encode the counters into clock values, an *N-wrapping* construction from [HKPV98] is used. All clocks x stay within the interval $[0, N]$ for a constant N by resetting each clock x as soon as $x = N$ (*wrapping reset*). For a dedicated system clock x_{sys} these are the only resets (which makes x_{sys} periodic). In this way, *wrapping values* for all other clocks can be defined as their values at the (periodic) times where $x_{sys} = 0$. Thus, between any two consecutive *non-wrapping resets* of the clock x (i.e., resets when $x < N$ holds), its wrapping value does not change.

Using this construction, the value v of a counter C can be kept as the wrapping value 2^{1-v} of a clock x_C. Therefore, this wrapping value of x_C is always smaller than or equal to 2. The conditional branching is done by directly comparing the value of x_C to 2, and the increment (decrement) operation is implemented as dividing (multiplying) the wrapping value of x_C by 2.

For the implementation of the decrement, i.e., the doubling of the value of x_C, a task Q with fixed computation time is released at a non-deterministically chosen time and a clock x_{Cnew} is reset at the same time. This task is then preempted by a task P of higher priority with variable execution time. P is

released when x_C is reset to zero by a wrapping edge. A guard with $x_{done} = 0$ is used to check if its execution time is equal to the wrapping value of x_C, i.e., if it finishes when x_{sys} is reset. This preemption is repeated, and by using the $x_{done} = 0$ guard again afterwards to check that Q finishes when x_{sys} is reset, the wrapping value of x_{Cnew} is forced to be the double of the wrapping value of x_C. The response time of Q is a constant time (its computation time) plus two times the wrapping value of x_C – because of the two preemptions from P. If any step in the construction fails (some non-deterministic choice was wrong), the automaton enters a sink location where no task is released. Figure 1 illustrates a successful run of the whole decrement procedure.

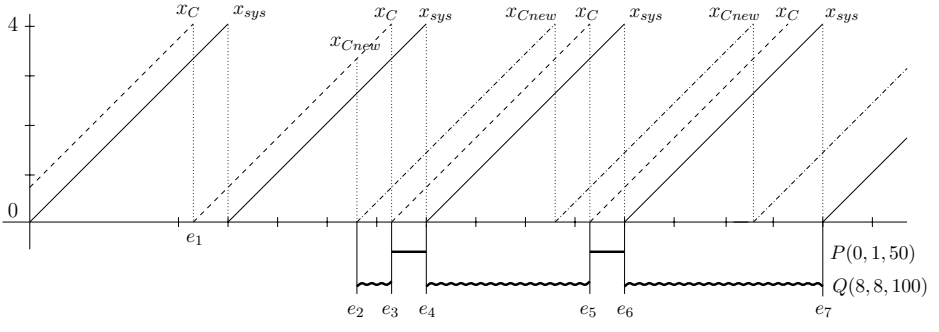


Fig. 1. Time chart of the doubling procedure using the *N-wrapping* construction

An increment for C needs to halve the wrapping value of the clock x_C. To achieve this, the wanted new value is simply guessed in a clock x_{Cnew} and then checked using the above decrement procedure. Only if the double of x_{Cnew} is x_C, the computation will continue.

During all three operations, all used tasks meet their deadline (using a fixed priority scheduler) and therefore the automaton is schedulable as long as it executes these instructions. A special location l_{halt} is used to represent the halt state of M. This location releases the task Q and it has an unguarded selfloop. Therefore, if A_M reaches l_{halt}, it will be able to release an unbounded amount of task instances at the same timepoint, making the system unschedulable.

Lemma 2 ([FKPY07]). *For a given two-counter machine M, the constructed task automaton A_M is unschedulable with the fixed priority scheduler if and only if M halts.*

Note that the construction uses all three properties (preemption, variable execution time, and feedback) given in Section 2.

4.2 Undecidability on Multi-processor Without Feedback

We extend this result to the multi-processor case for the systems with preemptive schedulers and variable computation time of tasks, but the finishing time of a

task is never tested in the guards and invariants of a task automaton, that is, task feedback is not allowed (x_{done} is never used in guards and invariants). To achieve the same effect as with the task feedback, we develop two mechanisms based on the status of task queues and the multi-processor scheduling policy. First, we describe how the constraints $x_{done} = 0$ can be replaced by certain task releases so that the system is still able to detect whether the task in question finished at the right time. Secondly, a construction will be given to store this information in a task queue.

Both mechanisms will ensure that the demand for the task feedback in the construction is removed. There is no branching upon the fact whether $x_{done} = 0$, the automaton continues its computation even if some non-deterministic choice of the system is wrong. It stores the information about the fact whether all non-deterministic choices have been correct so far and only if this is true then a queue can become unschedulable after entering the location l_{halt}. However, to store this information, at least one additional processor with its own queue is needed.

Checking the precise finishing time. The construction of A_M for a given two-counter machine M is exactly the same as before, but we change three aspects. Note that we are now in the multi-processor scenario with at least two queues q_1 and q_2. The tasks P and Q used in the construction above will be scheduled to q_1, P always preempts Q.

To check the finishing times of P and Q, two tasks T^P_{chk1} and T^P_{chk2} for P (T^Q_{chk1} and T^Q_{chk2} for Q) are released at the same timepoint (when we expect P or Q to finish). From now on, we will talk only about checking of the correct finishing time of the task P. All mechanisms are analogous for the task Q. The scheduler then detects, if the task P finishes (is removed from the queue) between both task releases. Future decisions of the scheduler (like scheduling tasks in the halt location in an schedulable or unschedulable order) can be based on this. The construction works in detail as follows.

First, at each position where an edge contains the guard $x_{done} = 0$ for synchronizing on the finishing of a task P, remove the guard and add two additional locations l_1 and l_2. The first one releases T^P_{chk1}, the second one T^P_{chk2}. Both released instances will be scheduled to the queue q_2. The original edge which contained $x_{done} = 0$ will go to l_1. The edges from l_1 to l_2 and from l_2 to the old location will contain guards ensuring that the automaton does not delay in l_1 and l_2. The whole fragment is depicted in Figure 2.

Secondly, the scheduler can use the task releases of T^P_{chk1} and T^P_{chk2} to check if P stayed in the queue until that timepoint, but not longer. If P *is* in the queue at the release of the first "checking task" T^P_{chk1} and if it *is not* in the queue when the second "checking task" T^P_{chk2} is released, the scheduler knows that it has finished at the correct timepoint. If it is not the case, then the automaton cannot guarantee that P has finished at the correct timepoint and thus it cannot guarantee that the simulation of the two-counter machine is correct.

Using the queue as a memory cell. The remaining problem is to remember the information that all finishing times were correct so far. Note, that it is

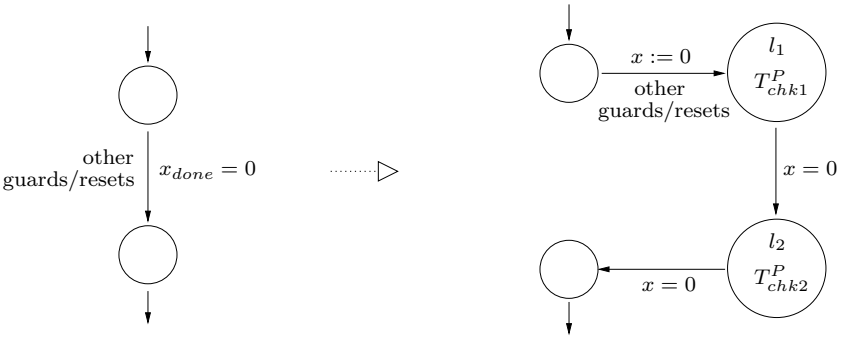


Fig. 2. Replacing the check for $x_{done} = 0$ with two additional locations l_1 and l_2 and an additional clock x

impossible to send this information back to the automaton and that the scheduling function is stateless. However, we now describe how the second queue q_2 can be used to remember this.

Except for T^P_{chk1} and T^P_{chk2}, there will be also tasks of an additional type T_{mark} released by the automaton and put into q_2 in such a way that the utilization of the second processor is 100%. Directly before (and at the same time of) the release of T^P_{chk1}, an instance of T_{mark} is released and put to the end of q_2. If the following release of T^P_{chk1} detects an early finishing (P is not in the queue q_1), the scheduler puts T^P_{chk1} directly *after* T_{mark} in q_2, otherwise directly *before* T_{mark}. The same holds for T^P_{chk2}, if the scheduler detects a late task finishing (P is still in q_1). In this way, the fact that M has been simulated correctly so far is equivalent to the fact that the last task in q_2 is of type T_{mark}. Such a successful scenario is described in Figure 3.

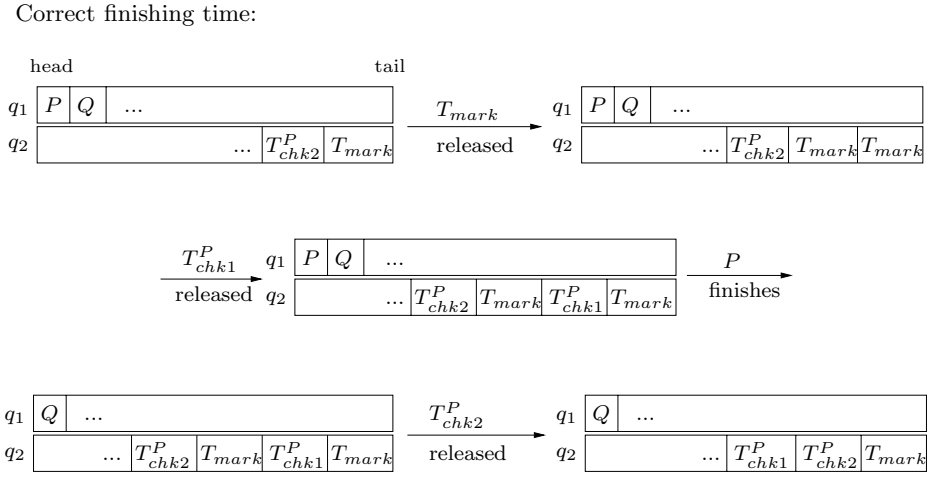


Fig. 3. Releases of tasks T^P_{chk1}, T^P_{chk2}, and T_{mark} detect that the finishing time of P is correct

In case that this checking discovered that the simulation was not correct (the last task in q_2 is not T_{mark} as the result of the checking procedure), we want to remember this information for the rest of the computation. We encode it by the fact that after any further checking, the last task in q_2 will not be T_{mark}. In the following, we explain how this information gets propagated from one checking to another.

All task types T^P_{chk1}, T^P_{chk2} and T_{mark} have fixed computation times. Since the checking times for each instruction are known in advance, it is possible to adjust these computation times so that the processor is never idle, but also no deadline is missed. Therefore, when new tasks T^P_{chk1}, T^P_{chk2} and T_{mark} arrive to the queue, the task instance from the previous checking which was scheduled as last is still in the queue. The scheduler can then take into consideration whether it is T_{mark}. If it is the case, the scheduler inserts the new tasks into q_2 as described above. Otherwise, the scheduler just makes sure that the last task is not T_{mark} again, keeping this fact invariantly true until the end of the computation. Therefore, A_M does not cheat during the whole computation (all task instances finish when we wanted them to finish) if and only if all checks with T^P_{chk1} and T^P_{chk2} invocations are successful which is equivalent to the fact that the last task in q_2 is T_{mark}.

The last thing which is changed from the single-processor case with feedback is the halt location. Here, instead of a selfloop releasing unboundedly many task instances, two tasks $R_1(1,1,1)$ and $R_2(1,1,2)$ are released at the same time, and then the automaton enters a sink location where no tasks are released anymore, as depicted in Figure 4.

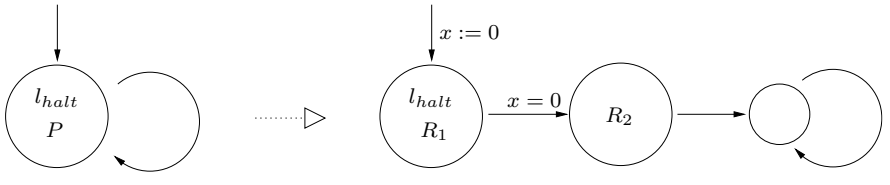


Fig. 4. The new encoding of the halt state

The scheduler puts both instances to the head of the same queue and lets R_2 be computed first, making R_1 miss its deadline, if and only if A_M simulated M correctly, i.e., the last task in q_2 is T_{mark}. In this way, the automaton A_M will only be unschedulable, if it can reach the location l_{halt} without cheating at the task finishing transitions. The following lemma states the correctness of the construction.

Lemma 3. *For a given two-counter machine M, the task automaton A_M is not schedulable with the constructed multi-processor scheduler if and only if M halts.*

Proof (Sketch). If the two-counter machine M reaches the halt state then A_M can simulate the same instructions leading to the halt location. During the whole (correct) simulation, the executions of P and Q always finish at the correct

timepoint, i.e., when x_{sys} is reset (and when x_{done} was reset in the construction with the feedback). This keeps the wrapping values of the clocks x_C and x_D correctly encoding the values of the counters C and D, respectively, and q_2 is in a "not cheated" state all the time (the last task in q_2 is T_{mark}). When A_M then enters its halt location, R_1 and R_2 will be executed in the order which makes the queue unschedulable.

If M does not reach its halt state then A_M has to cheat to reach its halt location, which means that the following holds for the task instances of P and Q. There is an instance of P or Q which does not finish at the time when x_{sys} is reset. Consequently, this cheating of A_M is detected by the scheduler through releases of T^P_{chk1} and T^P_{chk2}. From this timepoint on, the last task in q_2 will not be T_{mark}. This means that A_M can only reach the halt location with q_2 expressing that a task finished at a wrong time, making the scheduler execute R_1 before R_2, which means that both of them meet their deadlines. Provided that during the run up to this point all tasks met their deadlines (which is ensured by construction), the automaton A_M is schedulable. Also, no tasks miss their deadlines along runs of A_M which do not reach the halt location. □

5 Conclusion

Task automata, an extended version of timed automata with computation tasks, may serve as a general model for real-time systems with non-uniformly recurring tasks. In our previous work, the classic notion of schedulability has been extended to this model by possibility of specifying various scheduling policies by means of (a restricted class of) timed automata. The model includes many features such as variable computation time of tasks and preemptive schedulers. The decidability of the schedulability problem has been studied for various subclasses. However, all the results were shown only for the single-processor setting.

In this paper, we have defined this problem for the multi-processor systems by extending the definition of the scheduling policies. We have shown that the schedulability problem stays decidable in the multi-processor setting for the classes of task automata with a non-preemptive scheduling strategy or with fixed computation times of tasks. On the negative side, the problem turns out to be undecidable for preemptive multi-processor schedulers when the computation times of tasks may vary within an interval. It is still an open question, whether this problem is decidable in the single-processor setting. As a future work, we will try to close this decidability gap.

References

[ABJ01] Andersson, B., Baruah, S., Jonsson, J.: Static-priority scheduling on multiprocessors. In: Proc. of RTSS '01, pp. 193–202. IEEE Computer Society Press, Los Alamitos (2001)

[AD94] Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)

[BB06] Baker, T., Baruah, S.: Handbook of Real-Time and Embedded Systems. In: Schedulability Analysis of Multiprocessor Sporadic Task Systems, CRC Press (2006)

[BCL05] Bertogna, M., Cirinei, M., Lipari, G.: Improved schedulability analysis of EDF on multiprocessor platforms. In: Proc. of ECRTS '05, pp. 209–218. IEEE Computer Society Press, Los Alamitos (2005)

[But97] Buttazzo, G.C.: Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications. Kluwer Academic Publishers, Dordrecht (1997)

[EWY99] Ericsson, C., Wall, A., Yi, W.: Timed automata as task models for event-driven systems. In: Proc. of RTCSA'99, pp. 182–205. IEEE Computer Society, Los Alamitos (1999)

[FKPY07] Fersman, E., Krcal, P., Pettersson, P., Yi, W.: Task automata: Schedulability, decidability and undecidability. Information and Computation 205(8), 1149–1172 (2007)

[FMPY06] Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: Schedulability analysis of fixed priority systems using timed automata. Theoretical Computer Science 354(2), 301–317 (2006)

[HKPV98] Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? Journal of Computer and System Sciences 57, 94–124 (1998)

[LL73] Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20(1), 46–61 (1973)

[RC01] Real, J., Crespo, A.: Offsets for scheduling mode changes. In: Proc. of ECRTS'01, pp. 3–10. IEEE Computer Society Press, Los Alamitos (2001)

Designing Consistent Multimedia Documents: The RT-LOTOS Methodology

Paulo N.M. Sampaio[1], Laura M. Rodríguez Peralta[1], and Jean-Pierre Courtiat[2]

[1] Distributed Systems and Networks Lab. (Lab-SDR)
University of Madeira (UMA)
Campus da Penteada 9000-390
Funchal, Madeira, Portugal
Tel: +351 291 705291
psampaio@uma.pt
[2] LAAS – CNRS
7, Av. du Colonel Roche 31077
Toulouse, France
Tel: +33 5.61.33.62.44
courtiat@laas.fr

Abstract. Interactive Multimedia Documents (IMDs) are expected to satisfy temporal consistency properties in order to ensure that their synchronization constraints (temporal, logical and causal) can be respected during their presentation. If an inconsistent situation can not be detected previously, the presentation of the document can be lead to undesirable deadlocks (global or partial). In particular, the flexibility of high level authoring models (such as SMIL 2.0) for the edition of complex IMDs can lead authors, in certain cases, to specify inconsistent documents. For this reason, it is necessary to apply a methodology that provides the formal modelling for the dynamic behavior of the document, consistency checking, and the scheduling of the presentation taking into account the temporal non-determinism of these documents. This paper presents the main results of the development of a formal methodology which is based on the Formal Description Technique RT-LOTOS, and which has been successfully applied to support the design of complex SMIL 2.0 documents.

1 Introduction

The definition of Interactive Multimedia Documents (IMDs) is related to the coordinated presentation of different types of information (text, images, audio, video, etc.) possibly associated with user interactions. The quality of the presentation of these documents depends upon the global consistency of their temporal synchronization constraints. An IMD is considered consistent when all of its synchronization constraints can be respected during the presentation. Otherwise, if a synchronization constraint can not be respected, the presentation of the document can lead to a deadlock situation. Unfortunately, little has been done in order to propose an integrated solution for the verification of consistency properties, and scheduling of IMDs [1], [2], [3], [4], [5].

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 290–303, 2007.

Over the last few years, the consortium W3C proposed the language *Synchronized Multimedia Integrated Language (SMIL)* in order to provide the integration of Interactive Multimedia Documents on the web. SMIL has been largely applied for the design and presentation of these documents [6], [7], [8]. The second version of SMIL [9] was proposed not only to overcome the limitations of the first version of this language, but also to extend the functionalities of SMIL with the addition of animation, transition effects, etc. One important feature of SMIL 2.0 is its temporal model which was much more flexible allowing the author to specify synchronization constraints between any media object of a document. Although, the temporal model of SMIL 2.0 was much more expressive, it could lead authors to describe synchronization constraints that cannot be resolved consistently at document presentation time. For this reason, a formal methodology is necessary to support the design of SMIL 2.0 documents.

This paper presents the main results of the development of a methodology for the design, consistency analysis, scheduling and presentation of IMDs. This methodology presents a solid formal basis, the Formal Description Technique Real Time LOTOS (and its simulation/verification environment RTL – RT-LOTOS Laboratory which was developed at LAAS-CNRS [10]). The choice for RT-LOTOS was motivated by our research group´s solid previous experience with it and with the set of available tools (RTL). Besides, RT-LOTOS enables the complete management of the temporal non-determinism (e.g., related to the unknown instant of the end of presentation of a remote continuous media object, or to the occurrence of a user interaction) inherent to IMDs during their presentation. This paper also presents how this methodology was successfully applied to support the design of SMIL 2.0 documents enabling their consistent presentation on the web.

This paper is structured as follows: Section 2 presents a global view of the applied formal design methodology and briefly presents some characteristics of RT-LOTOS. Section 3 characterizes temporal inconsistencies and introduces a simple IMD to illustrate the presentation of the formal methodology throughout the paper. Section 4 introduces the approach proposed for the automatic translation from a SMIL 2.0 document into an RT-LOTOS specification. Section 5 presents the verification of IMDs. Section 6 outlines the main aspects about the scheduling of IMDs. Section 7 introduces the presentation of consistent SMIL 2.0 documents. Section 8 presents some related works. Finally, section 9 presents some conclusions of this work.

2 The RT-LOTOS Based Formal Design Methodology

The proposed methodology provides a framework for the design (specification, verification and presentation) of complex Interactive Multimedia Documents. Relying on the Formal Description Technique RT-LOTOS, and its associated simulation/ verification environment RTL, it yields three main contributions: it allows to define a formal semantics of the high-level document authoring model (SMIL 2.0), describing without any ambiguity the behavior of the document presentation; it allows to check consistency properties on the RT-LOTOS formal specification derived from the authoring model, using standard verification techniques (reachability analysis and model checking); it allows to generate automatically a scheduling automaton from the

underlying reachability graph, providing an easy way to automatically correct the document temporal structure by removing undesirable temporal inconsistencies.

This methodology is illustrated in Figure 1. Following the classical approach, a SMIL 2.0 document can be presented directly by a SMIL player, such as GRiNS [11], RealPlayer [12] or X-SMILES [13], among others. However, these tools do not provide the automatic detection and diagnosis of potential temporal inconsistencies. Contrarily, the RT-LOTOS based methodology represents a means to provide the complete diagnostic, correction and presentation of SMIL 2.0 documents. This methodology consists of the following phases: (1) edition of the IMD using SMIL 2.0 as the high-level authoring model, (2) automatic translation of logical and temporal structure of the document into an RT-LOTOS formal specification (the non-temporal components of the document are translated automatically into a contextual information description which is used as a support for its presentation), (3) derivation of a minimal reachability graph from the RT-LOTOS specification through the RTL tool [14], (4) verification of consistency properties by means of reachability analysis, and scheduling of the document presentation if there are valid (consistent) solutions for it by means of an operational scheduling graph, the TLSA [16].

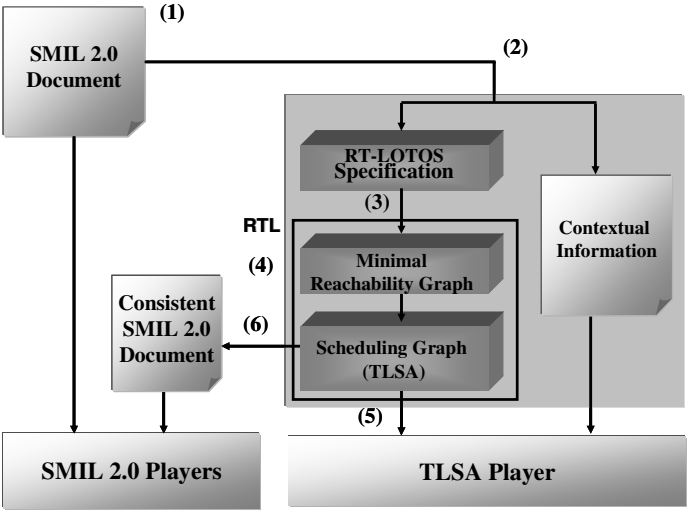


Fig. 1. Formal design methodology of SMIL 2.0 documents

Further on, two approaches are possible for presenting the resulting consistent document. Either a new consistent SMIL 2.0 document is generated from the TLSA, and the presentation may then be carried out using SMIL 2.0 players (6); or the presentation is carried out based on the TLSA and on the contextual information using a dedicated tool, the TLSA Player. The main results of the development of this formal methodology for the design of SMIL 2.0 documents are discussed on this paper.

3 Temporal Inconsistencies

Describing synchronization constraints in a complex IMD at authoring time can be an error-prone task, and some inconsistent scenarios can be created. Many reasons may potentially lead to temporal inconsistencies when an author uses a high-level and flexible authoring model, for instance: inconsistencies between the expected duration of the nodes and the logic of the synchronization links enforcing their termination, conflicting synchronization links, bad timing of the links, etc. In general, temporal inconsistencies are mostly the consequence of the occurrence of non-deterministic events (their occurrence can not be determined previously). These events can be [15]:

- *Internal non-deterministic events* which are related to the controlled adjustment of the presentation duration of a media (related to admissible QoS adjustments for the media), as well as to incomplete timing constraints;
- *External non-deterministic events* are related to the occurrence of external events, such as user interactions on anchors, network delays and processing results from data-base queries, scientific simulations, and so on.

Temporal inconsistencies may be the consequence of either internal or external non-determinism, or even both. The formal methodology presented in this paper can be applied to support the design of complex IMDs, although, in order to illustrate the application of this methodology for the verification of temporal inconsistencies in SMIL 2.0 documents a simple interactive multimedia scenario is applied and presented in Figure 2(a). An overview of the SMIL 2.0 document describing this scenario is presented in Figure 2(b).

The scenario describes the parallel presentation of an interactive image (*interactiveButton*) with the sequential composition of an exclusive presentation of two audio objects (*audio1* and *audio2*) followed by a video object (*video*). In this case, all the components of an exclusive presentation will be presented, however, never simultaneously, according to the semantics of this operator as described in [9]. The presentation duration of objects *audio1*, *audio2*, *video* and *interactiveButton* are respectively [0,5] seconds, [0,5] seconds, indefinite[1] and [0,20] seconds, and the following synchronization constraints are considered: (i) the presentation of the exclusive composition of *audio1* and *audio2*, and *interactiveButton* must start simultaneously, (ii) the presentation of *video* and *interactiveButton* must end simultaneously, (iii) if a user interaction (*activateEvent*) occurs during the presentation, the scenario is interrupted and then immediately restarted.

In this scenario, an inconsistency (desynchronization) can be produced whenever the presentation of *video* continues after the end of presentation of *interactiveButton* (Indeed, the author of the IMD mis-specified the previous synchronization constraints in SMIL by determining that the presentation of *video* can only be interrupted by the occurrence of a user interaction over *interactiveButton*, as described by the underlined synchronization constraint depicted in Figure 2b). This is clearly, an unexpected and undesired situation to the author of the document which can be easily detected in a small scenario, but could represent a drawback in a complex IMD. In this case, we

[1] An indefinite duration characterizes the presentation of an object inside the interval $[0,+\infty]$

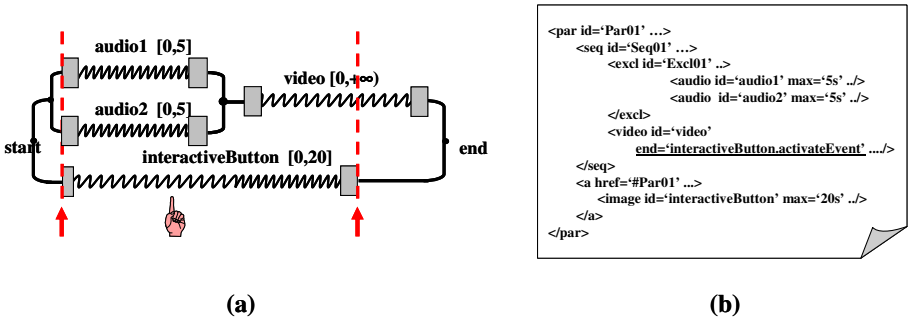


Fig. 2. Illustration of an interactive multimedia scenario

should note that the correct specification of the SMIL document depicted in Figure 2(b) would be <video id="video" **end="interactiveButton.endEvent"** …/>. The attribute *end* (in bold) states that the presentation of *video* always will be interrupted by the end of presentation of *interactiveButton* avoiding the occurrence of an unattended behaviour.

The main issues related to the application of the RT-LOTOS based design methodology to SMIL 2.0 documents are addressed in the next sections.

4 Translating SMIL 2.0 Documents into RT-LOTOS Specifications

The translation approach between SMIL and RT-LOTOS is based on the definition of intermediate structures representing all the components of the document and their temporal dynamics. This approach is structured into two main phases. In the first phase, two data structures - the *temporal tree* and the *synchronization arcs* - are derived from the SMIL document. The *temporal tree* describes hierarchically all the components of the document (media objects, containers like *par* - parallel composition - or *seq* - sequential composition - or *excl* - exclusive composition -, anchors, etc.) and their temporal characteristics. The *synchronization arcs* describe the causal relations among the components; they are expressed by *condition-action* relations. A *condition* is related to the occurrence (possibly associated with a time constraint) of either the beginning or the end of a component presentation, or a user interaction. An *action* is related to the execution of the beginning or the end of a component. In this phase a Contextual Information File (CIF) is also automatically generated from the SMIL document. The CIF describes the elements and attributes of each media object related to the spatial synchronization (height, length, etc.), content (url), presentation (volume) and the attributes describing the exclusive presentation of the components, animation, etc.

In the second phase, two additional data structures, called *ProcRTL* and the *synchronization points*, are derived from the previous data structures in order to automatically generate the RT-LOTOS processes. *ProcRTL* describes the structure of each RT-LOTOS process (process Id, behavior type referring to a library of predefined processes, observable gates, hidden gates, sub-processes, etc.). *Synchronization points* define the synchronization relations to be applied when composing the RT-LOTOS

processes. Due to space limitation we can not present examples with the RT-LOTOS specification associated with scenario of Figure 2.

This approach supports the complete mapping between SMIL 2.0 and RT-LOTOS and is effective to describe all the components and their temporal/logical behavior of complex IMDs. Once the RT-LOTOS specification related to the previous SMIL document is generated, some verification techniques can be applied in order to detect potential inconsistencies situation on the document's temporal structure.

5 Verification of Temporal Consistency of IMDs

The main goal of verifying the temporal consistencies of an IMD are: (i) to determine if the document can be presented correctly (if there are potential errors on the temporal structure of the document or not), (ii) if there are errors on the document, to identify which non-deterministic events are associated with these errors, and, (iii) to provide a proper diagnostic to the author so that he is able to correct the document.

For this purpose, some consistency properties were identified and they are further verified on the reachability graph that is automatically derived (by means of the RTL environment) from the RT-LOTOS specification related to the document. The reachability graph derived from the RT-LOTOS associated with the scenario presented in Figure 2 is depicted in Figure 3.

A reachability graph is a labelled transition system where each node (also called a class) of this graph corresponds to a control state and a clock region, and each arc corresponds to a labelled action. This labelled action can be the beginning (*start*) and end (*end*) of presentation of the document as a whole, the beginning (e.g., *sInteractiveButton*, *sVideo*, *excl_sAudio1_sAudio2*) and end (e.g., *eInteractiveButton_eVideo*, *excl_eAudio1_eAudio2*) of the presentation of media objects, user interactions (*activateEvent*), the progression of time (*t*), and some actions that are applied to support the presentation and hypermedia navigation (*triggerlink*, *linkExec*, *presentDoc* and *endRequest*). These actions are represented in the graph within an "i()" tag, such as *i(sVideo)*, since they are specified in RT-LOTOS as *internal actions* to the global behaviour of the document.

The basic idea of verifying the temporal consistency in a minimal reachability graph is to analyze the reachability of the action characterizing the end of presentation of the document (action *end*). Thus, considering that the reachability graph describes all the possible behaviors of the document, there are some paths that starting from the initial node of the reachability graph lead to the occurrence of the action *end*. These paths are called *consistent paths*. Contrarily, there can be some other paths that never will lead to the occurrence of the action *end*. These paths are called *inconsistent paths*. Based on the definition of consistent and inconsistent paths, the notion of consistency can be introduced for further verification using the reachability graph:

Definition 1: a document is *potentially consistent* if there is at least one consistent path on the reachability graph.

Definition 2: a document is *consistent* if all the paths of the reachability graph are consistent.

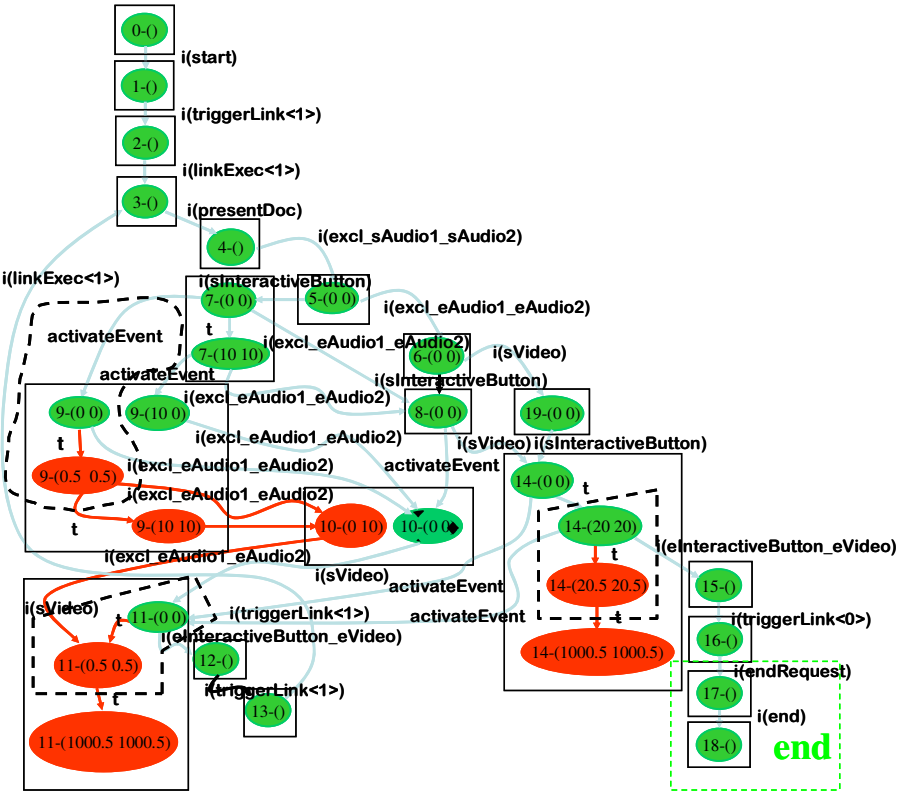


Fig. 3. Reachability graph associated with the scenario depicted in Figure 2

Definition 3: a document is *inconsistent* if there is no consistent path on the reachability graph. In other words, if all the paths of the graph are inconsistent. In this case, the document can not be presented and both its logical and temporal structures have to be reconsidered at authoring level.

For instance, by the verification of the consistency properties on the reachability graph associated with the previous IMD, the presentation of the document can be considered as *potential consistent* because some paths of this graph lead to the occurrence of the action end characterizing the end of presentation of the document. In this case, there are also some inconsistent paths on the reachability graph: (i) the inconsistent path that crosses the state 9-(0.5 0.5) which is related to the occurrence of an external non-deterministic event (a user interaction) during the exclusive presentation of *audio1* and *audio2*, and; (ii) the inconsistent paths that cross the states 11-(0.5 0.5) and 14-(20.5 20.5) that are related to the occurrence of an internal non-deterministic event (progression of time) since the presentation of *video* exceeds the global duration of the document presentation (20s). These inconsistent situations obviously do not respect the synchronization constraints of the document.

Note that each consistent path of the reachability graph corresponds to a potential scheduling solution for the document presentation. The purpose of scheduling the document presentation is to ensure that the document presentation follows a consistent path, and that, in the presence of several consistent paths, it follows the best consistent path with respect to some criteria. Thus, scheduling of the document can be carried out based on the results of the verification of consistency. The scheduling of an IMD is further discussed on the next section.

6 Scheduling the Presentation of Interactive Multimedia Documents

The reachability graph expresses all the possible behaviors for the presentation of the document (either consistent or inconsistent). However, this graph can not be applied for scheduling the presentation of a document since it does not provide clearly the temporal information related to the occurrence of each action. For this reason, to provide the scheduling of an IMD based on the results of the verification of consistency, all the inconsistent paths (generated either by internal or external non-deterministic events) must be discarded from the reachability graph to generate a consistent reachability graph. This consistent reachability graph provides the controllability of the document presentation determining all the valid temporal intervals inside which the presentation of the document will always be consistent.

Two issues have then to be addressed: (i) How to represent in a more compact and operational way all the consistent paths of the reachability graph. This point deals with the synthesis of a scheduling automaton (called a TLSA) from the consistent reachability graph (addressed in section 6.1), and; (ii) How to select among the consistent paths, the one which is the best with respect to some criteria. This point deals with the scheduling of an IMD based on a TLSA (addressed in section 6.2).

6.1 TLSA

A TLSA (Timed Labelled Scheduling Automata) features some specific characteristics, which differentiate it from traditional timed automata.

A TLSA has as many clocks as the number of states defined in the automaton. These clocks are called timers in order to distinguish them from the traditional clocks. Each timer measures the time spent by the automaton in the associated control state. No explicit function defines when a timer must be initialized. The timer associated with a control state is reset when the automaton enters this state, and its current value is frozen when the automaton leaves it (for this reason, the TLSA is considered as a single clock model, since, for any control state, there is only one active timer). Two temporal conditions are associated with each TLSA transition:

- A **firing window (indicated as W)**, which defines the temporal interval inside which the transition can be triggered. It is represented as an inequality to be satisfied by the timer associated with the current control state;
- An **enabling condition (indicated as K)**, which defines the temporal constraints to be satisfied in order to trigger the transition. It is represented as a conjunction of inequalities to be satisfied by a subset of timers, with exception of the timer associated with the current control state.

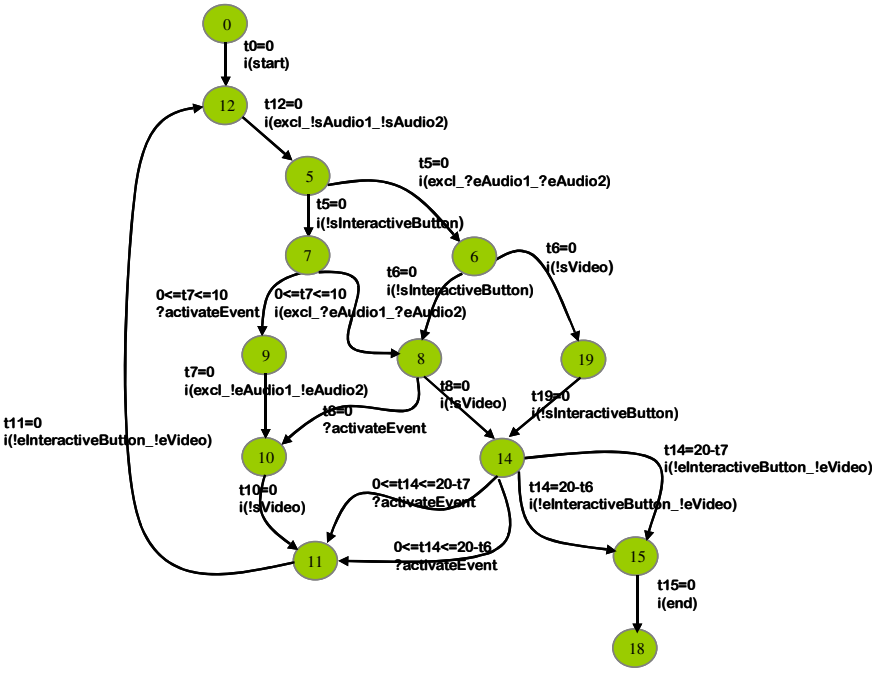


Fig. 4. TLSA associated with the scenario depicted in Figure 2

Intuitively, W characterizes the time during which the system can remain in the current control state. The enabling condition is related only to previous timers and thus characterizes the enabled transitions according to the behavior of the system. The formal semantics of the TLSA was defined in [16], and the algorithms for deriving automatically a TLSA from a consistent reachability graph were developed in [17].

The TLSA was proposed as a scheduling graph which allows the temporal formatting (the definition of valid temporal intervals for the presentation) of the document. However, some scheduling policies are needed in order to allow the orchestration of the document based on the TLSA. These scheduling policies are introduced in next section [18].

6.2 Scheduling an Interactive Multimedia Document Based on the TLSA

The TLSA allows the scheduling of an IMD based on timers which measure the time the system "spent" in each state of the automaton and which is also used to determine the triggering condition of a transition. Moreover, each transition of the TLSA is associated with the actions that can be handled by the presentation system such as starting or ending a presentation, announcing the occurrence of a user interaction, etc. Thus, the scheduling of an IMD using a TLSA can be carried out based on the definition of some important issues, such as handling *active* and *passive* actions.

An action is called **active** (represented by !) if its occurrence does not depend on its environment (e.g., the end of presentation of an image decided by the presentation system). Thus, active actions may be related to start and end actions of all media

objects (e.g., video, audio, text, image, etc.). These actions can be generated as soon as the scheduler decides when to trigger them based on their associated temporal firing window (W) specified in the TLSA.

An action is called **passive** (represented by ?) if its occurrence does depend on the environment of the presentation system (e.g., a user interaction, or the end of the presentation of a remote continuous media). The occurrence of passive actions cannot be predicted, but can be controlled within their valid temporal firing window in the TLSA. Thus, the scheduler waits until the environment triggers these actions and ensures that they are triggered inside their temporal firing window.

The notion of active and passive actions enables to determine scheduling policies for the execution of all the actions for the presentation of a document. Let thus W = [min,max] be the firing window associated with transition t, τ the firing time of transition t, and a the action labeling transition t.

- If *a* is active (e.g., *!sInteractiveButton*), then τ = min. In this case, this action is executed as soon as possible by the scheduler.
- if *a* is passive (e.g., *?excl_eAudio1_Audio2*), then $\tau \in$ [min, max]. In this case, the scheduler waits for the occurrence of the action, and ensures it takes place within the temporal interval [min, max]. In certain cases (e.g., related to the end of presentation of remote continuous media), the scheduler (implemented by the Player) may force at τ = max the execution of a passive action or the firing of an exception action..

The TLSA associated with the scenario of Figure 2 is presented in Figure 4. Each node of this TLSA represents a control state and each transition is described by a firing window (W) and an action (active or passive). For instance, when the scheduler enter the state 7, it has two execution possibilities within [0,10] seconds, (i) waiting for a user interaction to take place (passive action), before following the transition 7-9, or (ii) waiting for the end of the exclusive presentation of *Audio1* and *Audio2* (also a passive action), before following the transition 7-8. If none of these events take place after 10 seconds, the scheduler forces the end of the exclusive presentation. With the definition of the scheduling policies with the active and passive actions, the presentation of an IMD can be carried out.

7 Presenting Consistent IMDs

A prototype for the presentation of IMDs, called TLSA Player, was implemented based on the scheduling policies presented previously. The TLSA Player relies on the TLSA and on the contextual information for supporting the presentation of complex and consistent IMDs. The TLSA Player was implemented using JAVA (jdk 1.2) [19] and JMF 2.0 [20]. Figure 5 illustrates a snapshot of the TLSA Player.

The main advantage of using the TLSA for controlling the presentation of a document is that it provides the definition of the temporal intervals inside which the execution of this document will always be consistent. This is the main reason why the TLSA is the basis for the proposal of the approach for the presentation of an IMD where indeed all of its execution solutions are consistent.

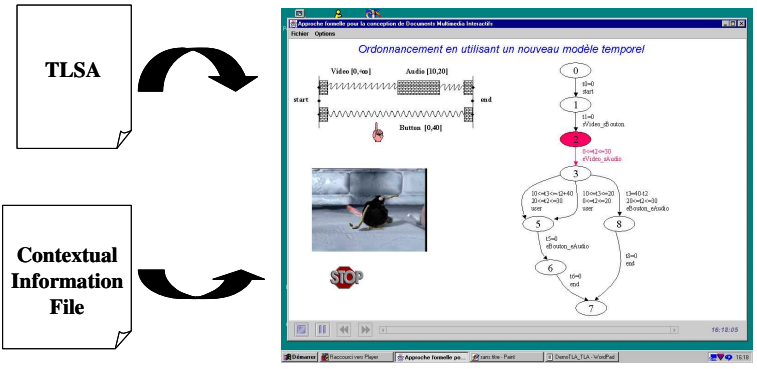


Fig. 5. Snapshot of the TLSA Player

The implementation of the TLSA Player allowed the presentation of complex Interactive Multimedia Documents based on the verification and scheduling techniques using the reachability graph and TLSA. This tool represents a straightforward solution in order to present consistent complex IMDs and to validate the proposed formal methodology for the design of IMDs.

8 Related Works

The design of IMDs has been addressed in the literature by different approaches in order to deal with issues such as temporal structure modeling, temporal flexibility, consistency checking, scheduling, and presentation based on optimization techniques.

The logical and temporal behaviors of a document can be usually described using an internal structure which enables to set up techniques for consistency checking and scheduling. A current solution for consistency checking consists in using a temporal graph (such as a directed acyclic graph) and specific algorithms (such as shortest path solution, or based on linear programming) to determine valid solutions for the presentation of a document. Some approaches which carry out consistency checking on multimedia documents are: constraint-based graphs (CHIMP [21], MADEUS [22], ISIS [23], IMAP [24], MPGS [25], using directed graphs [26], and using Temporal Access Control operators [3]), based on transition systems (TOCPN [27]), and based on the state of the components of document (FLIPS [28]). Further on, the presentation of these documents can be scheduled using different approaches as well, such as: timeline (GRiNS [29]), linear programming (Firefly [30]), constraint-based graphs (Tiempo [31], MAVA [32], and directed graphs [26]), transition systems (Dynamic Extended Finit State Machines - DEFSMs [33]), and partial order associated with linear programming and constraint-based graphs [34].

During scheduling and presentation of the document, some optimization techniques can also be used in order to avoid the degradation of the quality of the presentation. These techniques propose either alternatives execution planning so that different

configurations of the scenario are presented according to the resources available of the system and to the QoS parameters [31], or the adoption of a predictive solution in order to pre-fetch in memory the media objects that must be presented [35].

Some efforts have been made for analyzing consistency properties during the design of IMDs. However, few has been done for the formal modeling, verification and scheduling of consistent SMIL documents [36], [37], [38], [18]. In general, the works presented in [36, 37] propose a formal semantics for SMIL documents, but do not introduce any verification or scheduling approaches. The work presented in [38] proposed the verification of the main sources of inconsistencies in a SMIL 2.0 document; however this work did not propose an appropriate automatic integrated verification and scheduling approach for consistent solutions to an IMD.

In general, most of the approaches in the literature do not provide an integrated solution (e.g., verification, scheduling, controllability and optimization techniques) for supporting the design of complex IMDs. Contrarily, the methodology based on the Formal Description Technique RT-LOTOS offers an integrated and reliable framework, and relies on a transition system derived from the RT-LOTOS specification for further consistency checking, scheduling and presentation of consistent IMDs. The main advantages of this methodology based on RT-LOTOS are: (i) it offers a great authoring flexibility since it can be easily adaptable to any high-level authoring model; (ii) it allows for the verification, scheduling and managing the temporal non-determinism during the presentation of the document (run-time controllability), and; (iii) it also applies an optimization technique for the predictive pre-fetching of media objects during the presentation of a document [18]. In particular, this methodology was successfully applied to support the formal design of SMIL 2.0 documents.

9 Conclusions

This paper presented the main contributions of methodology developed to support the formal design of Interactive Multimedia Documents (IMDs) based on the Formal Description Technique (FDT) RT-LOTOS. The main advantages of this methodology are (i) that it is not dependent upon a particular high level authoring model and (ii) that the FDT RT-LOTOS is used implicitly for the author of the document (what makes the approach more accessible and intuitive). Thus, the author designs his document using the high level language (or the paradigm) of his preference, and then the temporal and logical behavior of his document is translated into an RT-LOTOS specification. The compositional characteristic of the RT-LOTOS specifications enables to model complex temporal scenarios by the successive composition of simpler behaviors. Since RT-LOTOS has a formal semantics, it is also possible to carry out the verification of properties of temporal consistency, and to carry out the scheduling of the document based on a timed automata model (TLSA) derived from the subjacent RT-LOTOS specification. To illustrate the utilization of this methodology, the language SMIL 2.0 was adopted for the modeling of complex Interactive Multimedia Documents.

References

1. Courtiat, J.-P., de Oliveira, R.C.: Proving temporal consistency in a new multimedia synchronization model. In: Proc. of ACM Multimedia'96, Boston, USA (November 1996)
2. Layaïda, N., Sabry-Ismail, L.: Maintaining Temporal Consistency of Multimedia Documents Using Constraints Networks. In: Proceedings of the 1996 Multimedia Computing and Networking, San-José, USA, pp. 124–135 (February 1996)
3. Mirbel, I., Pernici, B., Sellis, T., Tserkezoglou, S., Vazirgiannis, M.: Checking temporal integrity of interactive multimedia documents. VLDB Journal 9(2) (2000)
4. Buchanan, M.C., Zellweger, P.T.: Automatic Temporal Layout Mechanisms Revisited. ACM Transactions on Multimedia Computing, Communications and Applications 1(1), 60–88 (2005)
5. Bulterman, D.C.A., Hardman, L.: Structured Multimedia Authoring. ACM Transactions on Multimedia Computing, Communications and Applications 1(1), 89–109 (2005)
6. Rutledge, L.: SMIL: 2.0 – XML for Web Multimedia. IEEE Internet Computing, pp. 78–84 (September-October 2001)
7. Bulterman, D.C.A.: SMIL: 2.0 – Part 1: Overview, Concepts and Structure. IEEE Multimedia, pp. 82–88 (October- December 2001)
8. Tae-Hyun, I., et al.: Simple and Consistent SMIL Authoring: No more editing and no more error. In: IEEE Multimedia Computing on the World Wide Web (2000)
9. SMIL 2.0 - Synchronized Multimedia Integrated Language. URL: http://www.w3.org/AudioVideo/
10. Courtiat, J.P., Santos, C.A.S., Lohr, C., Outtaj, B.: Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique. In: Computer Communications (July 23, 2000)
11. GRiNS–SMIL: 2.0 Player Home Page. URL: http://www.oratrix.com/GRiNS/SMIL2.0/
12. RealPlayer - Real Networks (2004), URL: http//www.real.com/player/
13. Xsmiles - X-Smiles Home Page (Version 0.5). URL: http://www.x-smiles.org/index.html
14. RTL - RT-LOTOS Laboratory. http://www.laas.fr/RT-LOTOS
15. Santos, C.A.S., Sampaio, P.N.M., Courtiat, J.-P.: Revisiting the concept of hypermedia documents consistency. In: 7th ACM International Conference on Multimedia (ACM Multimedia'99). Orlando, USA (November 1999) (short paper)
16. Lohr, C., Courtiat, J.-P.: From the specification to the scheduling of time-dependent systems. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 129–145. Springer, Heidelberg (2002)
17. Lohr, C.: Contribution à la conception de systèmes temps-réel s'appuyant sur la technique de description formelle RT-LOTOS. PhD Dissertation (in french), Université Paul Sabatier, Toulouse (December 2002)
18. Sampaio, P.: Conception formelle de documents multimédia interactifs: une approche s'appuyant sur RT-LOTOS. PhD Dissertation (in french), Université Paul Sabatier, Toulouse (April 2003)
19. JAVA - The Source for JAVA Technologies. URL: http://java.sun.com/
20. JMF - Java Media Framework 2.0 API home page. URL: http://java.sun.com/products/javamedia/jmf/
21. Candan, K.S., Prabhakaran, B., Subrahmanian, V.S.: CHIMP: A Framework for Supporting Distributed Multimedia Document Authoring and Presentation. In: Proc. of the ACM Multimedia, Boston, USA, pp. 329–339 (November 1996)

22. Jourdan, M., Ladayia, N., Roisin, C., Sabry-Ismail, L.: An Integrated Authoring and Presentation Environment for Interactive Multimedia Documents. In: Proc. of Int. Conf. on MultiMedia Modeling - MMM'97, World Scientific, Singapore (1997)
23. Song, J., Ramalingam, G., Miller, R., Yi, B.: Interactive authoring of multimedia documents in a constraint-based authoring system. Multimedia Systems 7, 424–437 (1999)
24. Vazirgiannis, M.: Interactive Multimedia Documents. LNCS, vol. 1564. Springer, Heidelberg (1999)
25. Bertino, E., Ferrari, E., Stolf, M., MPGS,: An Interactive Tool for the Specification and Generation of Multimedia Presentations. IEEE Transactions on Knowledge Data Enginering 12(1), 102–125 (2000)
26. Huadong, M., Shin, K.G.: Checking Consistency in Multimedia Synchronization Constraints. IEEE Transactions on Multimedia 6(4), 565–574 (2004)
27. Yoon, K., Berra, P.B.: TOCPN: Interactive Temporal Model for Interactive Multimedia Documents. In: International Workshop on Multimedia Database Management Systems (IW-MMDBMS'98), Dayton, Ohio, USA (1998)
28. Schnepf, J., Konstan, J.A., Du, D.: Doing FLIPS: FLexible Interactive Presentation Synchronization. Distributed Multimedia Center, Department of Computer Sciences, University of Minnesota (1996), ftp://ftp.cs.umn.edu/users/du/papers/flips.ps
29. GRiNSComposer - Oratrix Development BV. SMIL 2.0 Evaluation KIT. Online. Available: http://www2.oratrix.nl/W3C/pressRelease2
30. Buchanan, M.C., Zellweger, P.T.: Automatically generating consistent schedules for multimedia documents. Multimedia System Journal 1(2), 55–67 (1993)
31. Wirag, S.: Specification and scheduling of adaptative multimedia documents. In: Technical report TR-1999-04, University of Stuttgart (January 1999)
32. Hauser, J.: Realization of an extensible multimedia document model. In: Multimedia '99 - Media Convergence: Models, Technologies and Applications, Wien, pp. 113–122. Springer-Verlag, Heidelberg (1999)
33. Huang, C.-M., Lin, C.-H., Wang, C.: Specifying and Executing Interactive Multimedia Presentations using the Formal Approach. proceedings of National Science Concil 23(4), 495–510 (1999)
34. Riviere, N., Bradin-Chezaviel, B., Valette, R.: Propagation de contraintes et ordonnancement de documents multimédias. Rapport LAAS No02044. In: 5ème Journée Doctorales Informatique et Réseaux (JDIR'2002), Toulouse (France), pp. 211–218 (Mars 4-6, 2002)
35. Jeong, T., Ham, J.W., Kim, S.J.A: Pre-Scheduling Mechanism for Multimedia Presentation Synchronization. In: proceedings of International Conference on Multimedia Computing and Systems (ICMCS'97), Ottawa, Canada, pp. 379–386 (1997)
36. Jourdan, M.A: formal semantics of SMIL: a Web standard to describe multimedia documents. Computer Standards and Interfaces 23, 439–455 (2001)
37. Chung, S.M., Pereira, A.L.: Time Petri Net Representation of SMIL. In: IEEE Multimedia, pp. 64–72 (January-March 2005)
38. Chefrour, D.: Vérification de la cohérence temporelle des documents multimédia SMIL. Rapport DEA, Université Joseph Fourier, Grenoble, France (June 2001)

AMT: A Property-Based Monitoring Tool for Analog Systems*

Dejan Nickovic and Oded Maler

Verimag, 2 Av. de Vignate, 38610 Gières, France
{Dejan.Nickovic,Oded.Maler}@imag.fr

Abstract. In this paper we describe AMT, a tool for monitoring temporal properties of continuous signals. We first introduce STL/PSL, a specification formalism based on the industrial standard language PSL and the real-time temporal logic MITL, extended with constructs that allow describing behaviors of real-valued variables. The tool automatically builds property observers from an STL/PSL specification and checks, in an *offline* or *incremental* fashion, whether simulation traces satisfy the property. The AMT tool is validated through a Flash memory case-study.

1 Introduction

The algorithmic verification field has been centered around the decision procedures for model-checking temporal logic formulae. Temporal logic [MP95] is a rigorous specification formalism used to describe desired behaviors of the system. A number of efficient algorithms for translating temporal logic formulae into corresponding automata have been developed [VW86, SB00, GPVW95, GO01], resulting in the success of logics such as LTL and CTL and their common integration into main verification tools. The temporal logic-based formalisms were adopted by the hardware industry with the standard PSL [HFE04] specification language.

In order to reason about *timed* systems, a number of real-time formalisms have been proposed, either as extensions of temporal logics (MTL [Koy90],MITL [AFH96], TCTL [Y97]) or regular expressions (*timed* regular expressions [ACM02]). However, unlike in the untimed case, there is no simple correspondence between these logics and timed automata [AD94] used in the timed verification tools.

The verification in the *continuous* domain was made possible with the advent of *hybrid automata* [MMP92] as a model for describing systems that have continuous dynamics with switches, and the algorithms for exploring their state-space. Although a lot of progress has been done recently [ADF+06], the scalability still remains a major issue for the exhaustive verification of hybrid systems, due to the explosion of the state space. Moreover, property-based verification of hybrid systems is only at its beginning [FGP06].

Hence, the preferred validation method for continuous systems remains simulation/testing. However, it has been noted that the specification element of verification can be

* This work was partially supported by the European Community project IST-2003-507219 PROSYD (Property-based System Design).

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 304–319, 2007.

exported to the simulation through property monitors. The essence of this approach is the automatic construction of an observer from the formula in the form of a program that can be interfaced with the simulator and alert the user if the property is violated by a simulation trace. This process is much more reliable and efficient than visual (graphical or textual) inspection of simulation traces, or manual construction of property monitors.

This procedure is called *lightweight verification*, where the property monitor checks whether a finite set of traces satisfy the property specification. In the framework of software *runtime verification*, temporal logic has been used as the specification language in a number of monitoring tools, including Temporal Rover (TR) [Dru00], FoCs [ABG+00], Java PathExplorer (JPaX) [HR01] and MaCS [KLS+02]. The extensions of temporal logics that deal with richer properties were also considered in monitoring tools such as LOLA [DSS+05].

In [MN04], we have introduced STL, a language for relating temporal behaviour of continuous signals via their *static abstractions* and a procedure for offline monitoring of specifications written in STL against continuous input traces. This paper extends the STL logic with an *analog layer* in which one can apply operations on continuous signals directly, as well as the *finitary interpretation* of the temporal operators in the spirit of PSL. The resulting logic is called STL/PSL. The original offline monitoring algorithm is extended to an *incremental* (semi-online) version. The main contribution of this paper is the implementation of a stand-alone Analog Monitoring Tool (AMT) which integrates results presented in [MN04] and this paper. Finally, a case-study on the behaviour of a FLASH memory cell is conducted in order to validate the performance of the tool.

The rest of the document is organized as follows: in Section 2, we introduce the STL/PSL logic along with its semantic domain. Section 3 discusses the offline property checking algorithm from [MN04] and presents its incremental extension. The AMT tool is presented in Section 4 and Section 5 describes the Flash memory case-study. Finally, in Section 6 we conclude with a discussion on the achievements and future work.

2 Signals and Their Temporal Logic

The specification of properties of continuous signals requires an adaptation of the semantic domain and the underlying logic.

2.1 Signals

Let the time domain $\mathbb{T}$ be the set $\mathbb{R}_{\geq 0}$ of non-negative real numbers. A finite length signal ξ over an abstract domain $\mathbb{D}$ is a partial function $\xi : \mathbb{T} \to \mathbb{D}$ whose domain of definition is $I = [0, r), r \in \mathbb{Q}_{>0}$. We say that the length of the signal ξ is r, and denote this fact by $|\xi| = r$. We use the notation $\xi[t] = \perp$ when $t \geq |\xi|$. In this paper, we restrict our attention to two particular types of signals, Boolean signals ξ_b with $\mathbb{D} = \mathbb{B}$, and continuous signals ξ_a with $\mathbb{D} = \mathbb{R}$.

We first present some signal properties that are independent of the signal domain. The *restriction* of a signal ξ to length d is defined as

$$\xi' = \langle \xi \rangle_d \text{ iff } \xi'[t] = \begin{cases} \xi[t] \text{ if } t < d \\ \perp \quad \text{otherwise} \end{cases}$$

The *concatenation* $\xi = \xi_1 \cdot \xi_2$ of two signals ξ_1 and ξ_2 defined over the intervals $[0, r_1)$ and $[0, r_2)$ is a signal over $[0, r_1 + r_2)$ defined as

$$\xi[t] = \begin{cases} \xi_1[t] & \text{if } t < r_1 \\ \xi_2[t - r_1] & \text{otherwise} \end{cases}$$

The *d-suffix* of a signal ξ is the signal $\xi' = d \backslash \xi$ obtained from ξ by removing the prefix $\langle \xi \rangle_d$ from ξ, that is,

$$\xi'[t] = \xi[t + d] \text{ for every } t \in [0, |\xi| - d)$$

The *Minkowski sum* and *difference* of two sets P_1 and P_2 are defined as

$$P_1 \oplus P_2 = \{x_1 + x_2 : x_1 \in P_1, x_2 \in P_2\}$$
$$P_1 \ominus P_2 = \{x_1 - x_2 : x_1 \in P_1, x_2 \in P_2\}.$$

Signals can also be combined and separated using the standard operations of *pairing* and *projection* defined as

$$\xi_1 \mid\mid \xi_2 = \xi_{12} \text{ if } \forall t \; \xi_{12}[t] = (\xi_1[t], \xi_2[t])$$
$$\xi_1 = \pi_1(\xi_{12}) \;\; \xi_2 = \pi_2(\xi_{12})$$

In particular, $\pi_p(\xi)$ will denote the projection of the signal ξ on the dimension with domain $\mathbb{B}$ that corresponds to the proposition p (and likewise $\pi_s(\xi)$ denotes projection of the signal ξ on the dimension with domain $\mathbb{R}$ corresponding to the continuous variable s).

Non-Zeno Boolean signals of finite length admit a finite representation called *interval covering* defined as a sequence of intervals $I_0 \cdot I_1 \cdot \ldots \cdot I_k$ such that the value of ξ_b is constant in every interval, $\xi_b(I_i) = \neg \xi_b(I_{i+1})$ for all $i \in [0, k-1]$, $\bigcup_{i=0}^{k} I_i = I$ and $I_i \cap I_j = \emptyset$ for every $i \neq j$. An interval I is said to be *positive* if $\xi_b(I) = \mathrm{T}$ and *negative* otherwise. An interval covering $\mathcal{I}$ is said to be *consistent* with a signal ξ_b if $\xi_b[t] = \xi_b[t']$ for every t, t' belonging to the same interval I_i. We denote by $\mathcal{I}_{\xi_b}$ the *minimal* interval covering consistent with a finite variability signal ξ_b.

Unlike Boolean signals, continuous signals do not admit an *exact finite representation*. However, numerical simulators usually produce a *finite* collection of sampling pairs $(t, \xi_a[t])$ with t ranging over some interval $[0, r) \subseteq \mathbb{T}$. This finite representation is in contrast to continuous signals defined as *ideal mathematical objects* consisting of an uncountable number of pairs $(t, \xi_a[t])$ for all $t \in [0, r)$. We adopt the approach of representing continuous signals of finite length by using a finite set of sampling points. The signal value at the missing time instants $t \in (t_i, t_{i+1})$ corresponds to the interpolation between sample points $(t_i, \xi_a[t_i])$ and $(t_{i+1}, \xi_a[t_{i+1}])$.

2.2 STL/PSL Specification Language

In this section we describe the STL/PSL logic, as an extension of MITL [AFH96] and STL [MN04] logics. We use a layered approach in the fashion of PSL [HFE04], with the *analog layer* allowing to reason about continuous signals and the *temporal layer* relating the temporal behavior of different input traces. The "communication" between

the two layers is done via *static abstractions* that partition the continuous state space according to the satisfaction of some inequality constraints on the continuous variables.

Since STL/PSL is targeted for specifying properties to be used for *lightweight verification* over *finite* traces, we adopt the finitary interpretation used in PSL, by defining *strong* and *weak* forms of the temporal operators. The strong form of an operator requires the terminating condition to occur before the end of the signal, while the weak form makes no such requirements. In PSL for example, `until!` and `until` represent the strong and the weak forms of the until operator, respectively.

The *analog layer* of STL/PSL is defined by the following grammar:

$$\phi :== s \mid \texttt{shift}(\phi, \texttt{k}) \mid \phi_1 \star \phi_2 \mid \phi \star \texttt{c} \mid \texttt{abs}(\phi)$$

where s belongs to a set $S = \{s_1, s_2, \ldots, s_n\}$ of continuous variables, $\star \in \{\texttt{+}, \texttt{-}, \texttt{*}\}$, $\texttt{c} \in \mathbb{Q}$ and $\texttt{k} \in \mathbb{Q}^+$. Note that the analog operators defined above are the ones currently supported by the AMT tool, but can be easily extended to new ones.

The semantics of the analog layer of STL/PSL is defined as an application of the analog operators to the input signal ξ:

$$\begin{array}{ll} s[t] & = \pi_s(\xi)[t] \\ \texttt{shift}(\phi, \texttt{k})[t] & = \phi[t + \texttt{k}] \\ (\phi_1 \star \phi_2)[t] & = \phi_1[t] \star \phi_2[t] \\ (\phi \star \texttt{c})[t] & = \phi[t] \star \texttt{c} \\ \texttt{abs}(\varphi)[t] & = \begin{cases} \phi[t] & \text{if } \phi[t] \geq 0 \\ -\phi[t] & \text{otherwise} \end{cases} \end{array}$$

The *temporal layer* of STL/PSL is defined as follows:

$$\begin{array}{c} \varphi :== p \mid \phi \circ \texttt{ c} \mid \texttt{not } \varphi \mid \varphi_1 \texttt{ or } \varphi_2 \mid \texttt{eventually! } \varphi \mid \\ \texttt{eventually![a:b] } \varphi \mid \texttt{eventually[a:b] } \varphi \mid \\ \varphi_1 \texttt{ until! } \varphi_2 \mid \varphi_1 \texttt{ until![a:b] } \varphi_2 \end{array}$$

where p belongs to a set $P = \{p_1, p_2, \ldots, p_n\}$ of propositional variables, $\texttt{a}, \texttt{b}, \texttt{c} \in \mathcal{Q}$ and $\circ \in \{\texttt{>}, \texttt{>=}, \texttt{<}, \texttt{<=}\}$. Note that we include explicitly in the syntax *weak* and *strong* versions of *eventually* operators[1].

The satisfaction relation $(\xi, t) \models \varphi$, indicating that signal ξ satisfies φ at time t is defined inductively as follows:

$$\begin{array}{ll} (\xi, t) \models p & \text{iff } \pi_p(\xi)[t] = \text{T} \\ (\xi, t) \models \phi \circ \texttt{ c} & \text{iff } \phi[t] \circ \texttt{c} \\ (\xi, t) \models \texttt{not } \varphi & \text{iff } (\xi, t) \not\models \varphi \\ (\xi, t) \models \varphi_1 \texttt{ or } \varphi_2 & \text{iff } (\xi, t) \models \varphi_1 \text{ or } (\xi, t) \models \varphi_2 \\ (\xi, t) \models \texttt{eventually! } \varphi & \text{iff } \exists t' \geq t \texttt{ st } t' < |\xi| \texttt{ and } (\xi, t') \models \varphi \\ (\xi, t) \models \texttt{eventually![a:b] } \varphi & \text{iff } \exists t' \in t \oplus [a, b] \texttt{ st } t' < |\xi| \texttt{ and } (\xi, t') \models \varphi \end{array}$$

[1] Untimed eventually exists only in its strong form. Weak eventually is trivially satisfied by any finite trace ξ.

$(\xi, t) \models \texttt{eventually[a:b]}\ \varphi$ iff $\exists t' \in t \oplus [a, b]$ $\texttt{st}\ t' \geq |\xi|$ $\texttt{or}\ (\xi, t') \models \varphi$
$(\xi, t) \models \varphi_1\ \texttt{until!}\ \varphi_2$ iff $\exists t' \geq t$ st $t' < |\xi|$ and $(\xi, t') \models \varphi_2$ and $\forall t'' \in [t, t']\ (\xi, t'') \models \varphi_1$
$(\xi, t) \models \varphi_1\ \texttt{until![a:b]}\ \varphi_2$ iff $\exists t' \in t \oplus [a, b]$ st $t' < |\xi|$ and $(\xi, t') \models \varphi_2$ and $\forall t'' \in [t, t']\ (\xi, t'') \models \varphi_1$

An STL/PSL specification φ_{prop} is an STL/PSL temporal formula. The signal ξ satisfies the specification φ_{prop}, denoted by $\xi \models \varphi_{\text{prop}}$, iff $(\xi, 0) \models \varphi_{\text{prop}}$. Note that our definition of the semantics of the *until* and timed *until* operators differs slightly from their conventional definition since it requires a time instant t where both $(\xi, t) \models \varphi_2$ and $(\xi, t) \models \varphi_1$. From the basic STL/PSL operators, one can define standard Boolean and temporal operators, namely *always* and *weak until*, as well as *weak* and *strong* forms of timed *always* operators.

A large part of analog design is based on comparing waveforms (signals) with some reference signal that specify a desired behavior. These notions are formalized using a distance function (metric) which quantifies numerically the resemblance of two signals. Mathematically speaking, a metric space is a pair (X, d) such that X is the domain and $d : X \times X \rightarrow \mathbb{R}_+$ is a function satisfying: $d(x, x) = 0$; $d(x, y) = d(y, x)$ and $d(x, y) + d(y, z) \geq d(x, z)$. There are many ways to define distance functions on waveforms, by taking the maximum of the pointwise distance at every time t, summing/integrating over the pointwise distance, etc. Once such a distance d is defined, it can be used to define distance-based logical operators of the form $d(\xi, \xi') < c$ for some positive constant c. Below we define three such operators, the first is based on the maximal pointwise distance while the two others are based on the metric defined in [KC06a] which "tolerates" large pointwise deviations between the two signals if they last for a time shorter than `t` and occur at most once every `T-t` units. As one can see these operators constitute a syntactic sugar as they can be expressed in STL/PSL.

$\texttt{distance}(\phi_1, \phi_2, \texttt{c}) = \texttt{abs}(\phi_1\texttt{-}\phi_2)\ \texttt{<= c}$
$\texttt{distance}(\phi_1, \phi_2, \texttt{c}, \texttt{t}, \texttt{T}) = \texttt{abs}(\phi_1\texttt{-}\phi_2)\ \texttt{> c -> eventually![<=t]}$ $\texttt{always[<=T-t](abs}\ (\phi_1\texttt{-}\phi_2)\ \texttt{<= c)}$
$\texttt{distance}(\varphi_1, \varphi_2, \texttt{t}, \texttt{T}) = (\varphi_1\ \texttt{xor}\ \varphi_2)\texttt{-> eventually![<=t]}$ $\texttt{always[<=T-t]}\ (\varphi_1\ \texttt{iff}\ \varphi_2)$

3 Checking STL/PSL Properties

In this section we describe two algorithms for checking STL/PSL properties. Both algorithms are based on a process that we call *marking*, namely determining truth value of each subformula at every time instant t. The marking is a doubly-recursive process going from the atomic propositions upward to the top formula, and, due to the nature of future temporal logic, from truth values at time t to truth values at time $t' \leq t$. The marking process terminates when the value of the top formula at time 0 is determined.

Offline marking: This procedure assumes that the multi-dimensional input signal ξ is already available, and the marking procedure is applied to the entire signal, propagating backward at once the values of subformulae, up to obtaining the truth value of the main formula.

Incremental marking: The incremental procedure updates the marking each time a new segment of the input signal is observed. It is useful in detecting early violation of an STL/PSL property and can be applied in parallel with the simulation process. It can also be used for monitoring real, rather than simulated systems.

The offline marking procedure takes as arguments a temporal STL/PSL specification φ_{prop} and the input signal ξ that we treat as a global data structure and do not pass it explicitly as an argument to the procedure. The algorithm computes, from the bottom-up, a signal $\chi_\psi(\xi)$ for each subformula ψ of φ_{prop}.[2] If ψ is a temporal STL/PSL formula φ, $\chi_\varphi(\xi)$ is called the *satisfaction signal*. This signal satisfies $\chi_\varphi(\xi)[t] = 1$ iff $(\xi, t) \models \varphi$. If ψ is a formula ϕ from the analog layer of STL/PSL, $\chi_\phi(\xi)$ is the result of applying the operator ϕ to the (continuous) signal ξ. Whenever the identity of ξ is clear from the context, we will use the shorthand notation χ_ψ.

The algorithm is decomposed into two methods OFFLINE-T and OFFLINE-A as shown in Algorithm 1, computing the χ_ψ corresponding to the formula ψ from the temporal and the analog layer of STL/PSL, respectively. The top level formula φ_{prop} is monitored by invoking OFFLINE-T(φ_{prop}).

Algorithm 1. OFFLINE-T and OFFLINE-A

```
input : STL/PSL Temporal Formula φ and signal ξ
switch φ do
    case p
        χ_φ := π_p(ξ);
    end
    case φ ∘ c
        OFFLINE-A (φ);
        χ_φ := COMBINE(∘c, χ_φ);
    end
    case OP1(φ1)
        OFFLINE-T (φ1);
        χ_φ := COMBINE(OP1, χ_φ1));
    end
    case OP2(φ1, φ2)
        OFFLINE-T (φ1, φ2);
        χ_φ := COMBINE(OP2, χ_φ1, χ_φ2));
    end
end
```

```
input : STL/PSL Analog Formula φ and signal ξ
switch φ do
    case s
        χ_φ := π_s(ξ);
    end
    case OP1(φ1)
        OFFLINE-A(φ1);
        χ_φ := COMBINE(OP1, χ_φ1);
    end
    case OP2(φ1, φ2)
        OFFLINE-A(φ1, φ2);
        χ_φ := COMBINE(OP2, χ_φ1, χ_φ2);
    end
end
```

Most of the work is done in the COMBINE procedure which takes one or two signals (possibly of different length) and computes from them a new signal based on the specific operation. The approach is based on [MN04] with some extensions to deal with both strong and weak operators. We illustrate the procedure on few representative operations:

$\chi_\varphi :=$ **COMBINE**$(\mathtt{or}, \chi_{\varphi_1}, \chi_{\varphi_2})$ For the disjunction we first construct a refined interval covering $\mathcal{I} = \{I_1, \ldots, I_k\}$ for $\chi_{\varphi_1} || \chi_{\varphi_2}$ so that the mutual values of both signals become uniform in every interval. Then we compute the disjunction intervalwise, that is, $\varphi(I_i) = \varphi_1(I_i) \vee \varphi_2(I_i)$. Finally we merge adjacent intervals having the same Boolean value to obtain the minimal interval covering $\mathcal{I}_{\chi_\varphi}$.

[2] The notation ψ is used whenever it is not important whether ψ is a temporal or an analog layer formula.

$\chi_\varphi := \textsc{Combine}(\texttt{eventually![a,b]}, \chi_{\varphi_1})$ For every positive interval $I \in \chi_{\varphi_1}$ we compute its *back shifting* $I \ominus [a,b] \cap \mathbb{T}$ and insert it to χ_φ. Overlapping positive intervals in χ_φ are merged to obtain a minimal consistent interval covering. In the process, all the negative intervals shorter than $b-a$ disappear.[3]

$\chi_\phi := \textsc{Combine}(\star, \chi_{\phi_1}, \chi_{\phi_2})$ For the pointwise arithmetic operations on continuous signals χ_{ϕ_1} and χ_{ϕ_2}, we first take the union of their sampling points and extend each signal to the new points by interpolation. The signal $\chi_{\phi_1 \star \phi_2}$ is computed by applying the pointwise arithmetic operation to each pair of corresponding sampling points. An example of the arithmetic operation is shown in Figure 1.

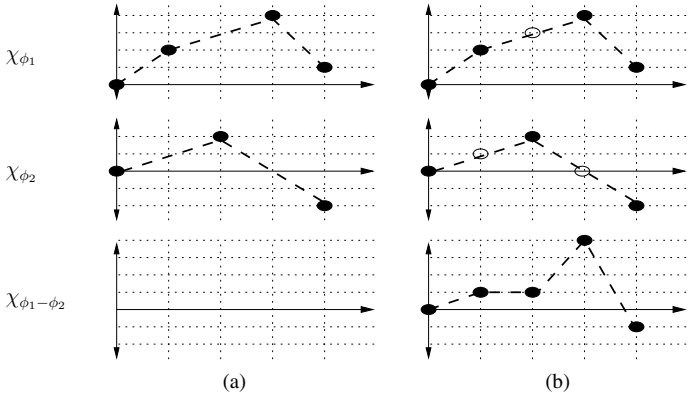


Fig. 1. Combining $\phi = \phi_1 - \phi_2$: (a) Input signals χ_{ϕ_1} and χ_{ϕ_2} sampled at different rates; (b) Refinement of χ_{ϕ_1} and χ_{ϕ_2} and computation of $\chi_{\phi_1-\phi 2}$

Incremental marking is performed using a kind of piecewise-online procedure invoked each time a new segment of ξ, denoted by Δ_ξ, is observed. For each subformula ψ the algorithm stores its already-computed associated signal partitioned into a concatenation of two signals $\chi_\psi \cdot \Delta_\psi$ with χ_ψ consisting of values already propagated to the super-formula of ψ, and Δ_ψ, consisting of values that have already been computed but which have not yet propagated to the super-formula and can still influence it.

Initially all signals are empty. Each time a new segment Δ_ξ is read, a recursive procedure similar to the offline one is invoked, which updates every χ_ψ and Δ_ψ from the bottom up. The difference with respect to the offline algorithm is that only segments of the signal that has not been propagated upwards participate in the update of their super-formulae. This may result in a considerable saving when the signal is very long.

As an illustration consider $\psi = \textsc{op}(\psi_1, \psi_2)$ and the corresponding truth signals of Figure 2-(a). Before the update we always have $|\chi_\psi \cdot \Delta_\psi| = |\chi_{\psi_1}| = |\chi_{\psi_2}|$: the parts Δ_{ψ_1} and Δ_{ψ_2} that may still affect ψ are those that start at the point from which the value of χ_ψ is still unknown. We apply Combine procedure on Δ_{ψ_1} and Δ_{ψ_2} to obtain a new (possibly empty) segment α_ψ of Δ_ψ. This segment is appended to Δ_ψ in order to be propagated upwards, but before that we need to shift the borderline between χ_{ψ_1}

[3] Another way to see it is as shifting the *negative* intervals by $[b,a]$.

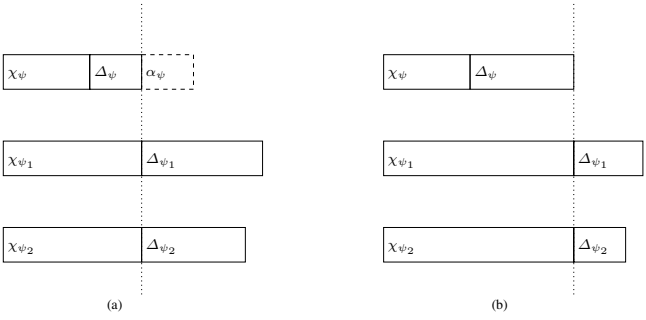


Fig. 2. A step in an incremental update: (a) A new segment α_ψ for ψ is computed from Δ_{ψ_1} and Δ_{ψ_2}; (b) α_ψ is appended to Δ_ψ and the endpoints of χ_{ψ_1} and χ_{ψ_1} are shifted forward accordingly

and Δ_{ψ_1} (as well as between χ_{ψ_2} and Δ_{ψ_2}) in order to reflect the update of Δ_ψ. The procedure is detailed in Algorithm 2.

Note that if $\chi_{\varphi_{\text{prop}}}$ becomes determined for time 0, the incremental procedure can be stopped. The finitary interpretation of temporal operators is used only if $\chi_{\varphi_{\text{prop}}}$ has not been determined after the end of simulation.

4 Overview of the AMT Tool

AMT is a stand-alone tool with a graphical user interface which implements the above algorithms with respect to sampled continuous signal inputs. AMT was written in C++ for GNU/Debian Linux x86 machines. The user interface is based on the library QT[4], while QWT[5] was used for visualizing plots.

The main window of the application is partitioned into five frames that allow the user to manage STL/PSL properties and input signals, evaluate the correctness of the simulation traces with respect to a specification and finally visualize the results. The **property edit** frame contains a text editor for writing, importing and exporting STL/PSL specifications, which are then translated into an internal data structure based on the parse-tree of the formula stored in the **property list** frame. An STL/PSL specification is imported into the **property evaluation** frame for its monitoring with respect to a set of input simulation traces, in either *offline* or *incremental* modes. The static import of the input traces is done via the **signal list** frame. The imported input signals, as well as signals associated to the subformulae of a specification can be visualized by the user from the **signal plots** frame. A screenshot of the main window is shown in Figure 3.

4.1 Property Management

The specifications in AMT are written in a simple editor with syntax highlighting for the extended STL/PSL language described below. An STL/PSL specification is then

[4] http://www.trolltech.com
[5] http://qwt.sourceforge.net

Algorithm 2. INCREMENTAL-T and INCREMENTAL-A

input : STL/PSL Temporal Formula φ and increment Δ_ξ
switch φ **do**
  case p
    $\Delta_\varphi := \Delta_\varphi \cdot \pi_p(\Delta_\xi)$;
  end
  case $\phi \circ c$
    INCREMENTAL-A (ϕ);
    $\alpha_\varphi :=$ COMBINE$(\circ c, \chi_\phi)$;
    $d := |\alpha_\varphi|$;
    $\Delta_\varphi := \Delta_\varphi \cdot \alpha_\varphi$;
    $\chi_\phi := \chi_\phi \cdot \langle \Delta_\phi \rangle_d$;
    $\Delta_\phi := d \backslash \Delta_\phi$;
  end
  case $\text{OP}_1(\varphi_1)$
    INCREMENTAL-T (φ_1);
    $\alpha_\varphi :=$ COMBINE$(\text{OP}_1, \chi_{\varphi_1}))$;
    $d := |\alpha_\varphi|$;
    $\Delta_\varphi := \Delta_\varphi \cdot \alpha_\varphi$;
    $\chi_{\varphi_1} := \chi_{\varphi_1} \cdot \langle \Delta_{\varphi_1} \rangle_d$;
    $\Delta_{\varphi_1} := d \backslash \Delta_{\varphi_1}$;
  end
  case $\text{OP}_2(\varphi_1, \varphi_2)$
    INCREMENTAL-T (φ_1, φ_2);
    $\alpha_\varphi :=$ COMBINE$(\text{OP}_2, \chi_{\varphi_1}, \chi_{\varphi_2}))$;
    $d := |\alpha_\varphi|$;
    $\Delta_\varphi := \Delta_\varphi \cdot \alpha_\varphi$;
    $\chi_{\varphi_1} := \chi_{\varphi_1} \cdot \langle \Delta_{\varphi_1} \rangle_d$;
    $\Delta_{\varphi_1} := d \backslash \Delta_{\varphi_1}$;
    $\chi_{\varphi_2} := \chi_{\varphi_2} \cdot \langle \Delta_{\varphi_2} \rangle_d$;
    $\Delta_{\varphi_2} := d \backslash \Delta_{\varphi_2}$
  end
end

input : STL/PSL Analog Formula ϕ and increment Δ_ξ
switch ϕ **do**
  case s
    $\Delta_\phi := \Delta_\phi \cdot \pi_s(\Delta_\xi)$;
  end
  case $\text{OP}_1(\phi_1)$
    INCREMENTAL-A (ϕ_1);
    $\alpha_\phi :=$ COMBINE$(\text{OP}_1, \chi_{\phi_1})$;
    $d := |\alpha_\phi|$;
    $\Delta_\phi := \Delta_\phi \cdot \alpha_\phi$;
    $\chi_{\phi_1} := \chi_{\phi_1} \cdot \langle \Delta_{\phi_1} \rangle_d$;
    $\Delta_{\phi_1} := d \backslash \Delta_{\phi_1}$;
  end
  case $\text{OP}_2(\phi_1, \phi_2)$
    INCREMENTAL-A (ϕ_1);
    INCREMENTAL-A (ϕ_2);
    $\alpha_\phi :=$ COMBINE$(\text{OP}_2, \chi_{\phi_1}, \chi_{\phi_2})$;
    $d := |\alpha_\phi|$;
    $\Delta_\phi := \Delta_\phi \cdot \alpha_\phi$;
    $\chi_{\phi_1} := \chi_{\phi_1} \cdot \langle \Delta_{\phi_1} \rangle_d$;
    $\Delta_{\phi_1} := d \backslash \Delta_{\phi_1}$;
    $\chi_{\phi_2} := \chi_{\phi_2} \cdot \langle \Delta_{\phi_2} \rangle_d$;
    $\Delta_{\phi_2} := d \backslash \Delta_{\phi_2}$;
  end
end

transformed into a structure adapted for the monitoring purpose, following the parse-tree of the formula. The user can hold more than one specification that is ready for evaluation in the property list frame.

Property Format. AMT tool extends the STL/PSL language described in Section 2.2 with additional constructs that simplify the process of property specification. Each top-level STL/PSL property is declared as an *assertion*, and a number of assertions can be grouped into a single logical unit in order to monitor them together at once. We also add a definition directive which allows the user to declare a formula and give it a name, and then refer to it as a variable within the assertions. The extended STL/PSL is defined with the following production rules

```
stl_psl_prop :==
  vprop NAME {
    { define_directive } { assert_directive }
}

define_directive :==
  define b:NAME := stl_psl_property
  | define a:NAME := analog_expression
```

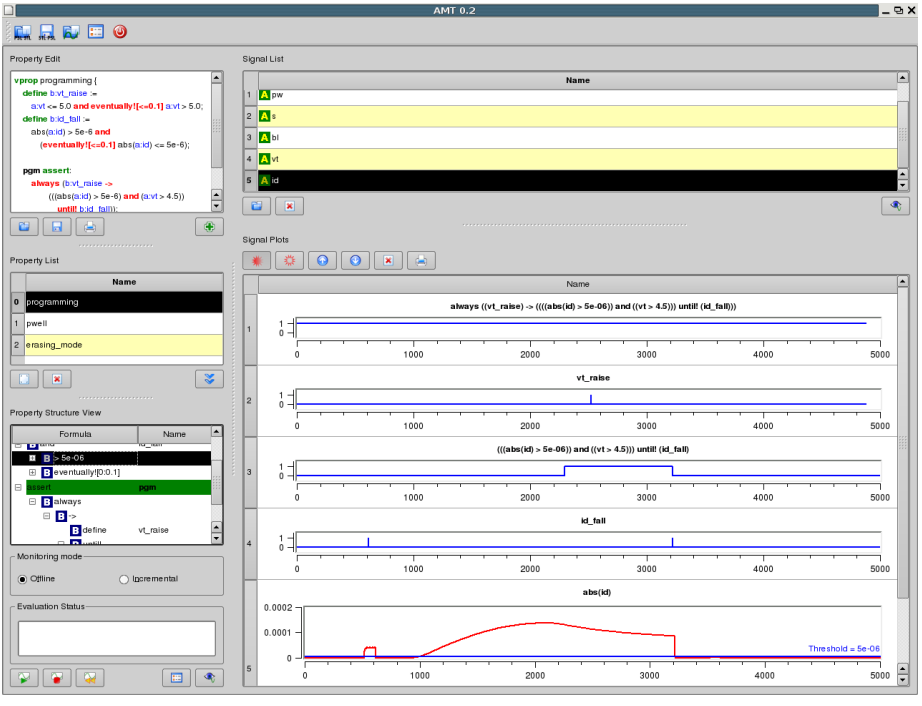


Fig. 3. AMT Main Window

```
assert_directive :==
  NAME assert : stl_psl_property
```

where `stl_psl_property` and `analog_expression` correspond to φ and ϕ from Section 2.2, respectively.

Property Evaluation. The correctness of an STL/PSL specification with respect to input traces is monitored through the property evaluation frame. The frame shows the set of assertions in a tree view, following the parse structure of the formula. The user can choose between *offline* and *incremental* evaluation of the specification.

In the offline case, the input signals are fetched from the signal list frame and the assertions are checked with respect to them. If one or more signals are missing, the monitoring procedure still tries to evaluate the property, but without guaranteeing a conclusive result.

For the incremental procedure, AMT acts as a server that waits for a connection from a simulator. Once the connection is established, the simulator sends input segments incrementally. The monitor alternates between reception of new input segments and incremental evaluation of the assertions. The user can configure the *timeout* value that defines the period between two consecutive evaluations. In between two such periods, the monitor accumulates input received from the simulator. There are three manners to end the incremental monitoring procedure: 1) All assertions become determined and

AMT stops the evaluation and closes the connection with the simulator; 2) The special termination packet is received from the simulator and 3) The user explicitly stops the procedure via the GUI.

AMT shows visually the evaluation result of an assertion, choosing a different color scheme for *undetermined*, *correct* and *incorrect* assertions. Each subformula of the specification has an associated signal with it, which can be visualized within the signal plots frame. The visualization of the associated signals can be used for understanding why an assertion holds/fails. During the incremental evaluation, all the signals within the signal plots frame are updated in real-time as new results are computed. The user can switch off the accumulation of intermediate results for better memory performance, thus discarding signals as soon as they are not needed anymore for the evaluation of super-formulae. In that case, the only output of the tool is the final answer.

4.2 Signal Management

The signals in AMT can be either continuous or Boolean. Signals are input traces that can be imported into the tool in an offline or incremental fashion. But signals are also associated to each subformula of an STL/PSL specification. The user can visualize them from the signal plots frame.

Offline Signal Input. Signals can be statically loaded from the signal list frame. Two file formats are currently supported by AMT:

out. The output format of the Nanosim simulations. The *current* and *voltage* signals are loaded, while *logical* signals are ignored.

vcd. The subset of Value Change Dump file format including real and 2-valued Boolean signals, commonly used for dumping simulations.

Incremental Signal Input. Signals can be imported incrementally to AMT, via a simple TCP/IP protocol. A simulator that produces input signals needs to connect to AMT during the *incremental evaluation* and send packets containing signal updates to the tool. The packets can be either Boolean or continuous signal updates, or a special *termination* packet, informing the tool that the simulation is over.

5 A FLASH Memory Case Study

The subject of the case study is the "Tricky" technology FLASH memory test chip in 0.13*us* process developed in ST Microelectronics Italy. The FLASH memory presents an advantage for the analog case study, in that it is a digital system whose logical behavior is implemented at the analog level. Hence, it is a good link between the analog and the digital world.

For the lightweight verification, the system under test is seen as a black box, and we do not need to know further details about the underneath chip architecture. The memory cell can be in one of the *programming*, *reading* or *erasing* modes. The correct functioning of the chip at the analog level in a given mode is determined by the behavior of a number of signals extracted during the simulation:

bl: matrix bit line terminal (cell drain)	**pw**: matrix p-well terminal (cell bulk)
wl: matrix word line (cell gate)	**s**: matrix source terminal (cell source)
vt: threshold voltage of cell	**id**: drain current of cell

The memory cell was simulated in the *programming* and the *erasing* modes for the case study, with the simulation time being 5000 *us* and 30000 *us* respectively. Four STL/PSL properties were written to describe the correct behavior of the cell in the *programming* mode and one property in the *erasing mode*. The AMT monitoring was done on a Pentium 4 HT 2.4GHz machine with 2Gb of memory. All the properties were found to be *correct* with respect to the input traces.

A detailed description of the properties and the monitoring results can be found in [NMF+06]. As an example, we consider the *erasing property*. The informal description of the property first defines the erasing condition, which is characterized by the wordline signal **wl** being lower than -6 and p-well **pw** above 5. Whenever the erasing condition holds, the pointwise distance between the source **s** and p-well **pw** voltages has to be smaller than 0.1 and the value of **pw** should not be greater than 0.83 from the value of bitline **bl**. The corresponding STL/PSL specification is:

```
vprop erasing {
  define b:erasing_cond :=
    a:wl <= -6 and a:pw > 5;

  erasing assert:
    always (b:erasing_cond ->
      (distance (a:s,a:pw,0.1)
      and (a:bl-a:pw)>-0.83));
}
```

Figure 4 shows some of the representative signals of the erasing property. We can mainly see that, whenever the *erasing condition* in Figure 4 (e) holds (denoted between two dashed lines), the pointwise distance between **s** and **pw** remains smaller than 0.1 (Figure 4 (h)) and the difference between **bl** and **pw** stays above the -0.83 threshold.

5.1 Tool Evaluation

The time and space requirements of AMT were studied with both *offline* and *incremental* algorithms. The complexity of the algorithm used in AMT is shown to be $O(k \cdot m)$ in [MN04] where k is the number of sub-formulae and m is the number of intervals.

Table 1 shows the size of the input signals (number of intervals). We can see that the *erasing* mode simulation generated 10 times larger inputs from the *programming* mode simulation. Table 2 shows the evaluation results for the *offline* procedure of the tool. Monitoring the properties for the programming mode required less than half a second. Only the `erasing` property took more than 2 seconds, as it was tested against a larger simulation trace. We can also see that the evaluation time is linear in the number of intervals generated by the procedure and can deduce that the procedure evaluates about 1.000.000 intervals per second.

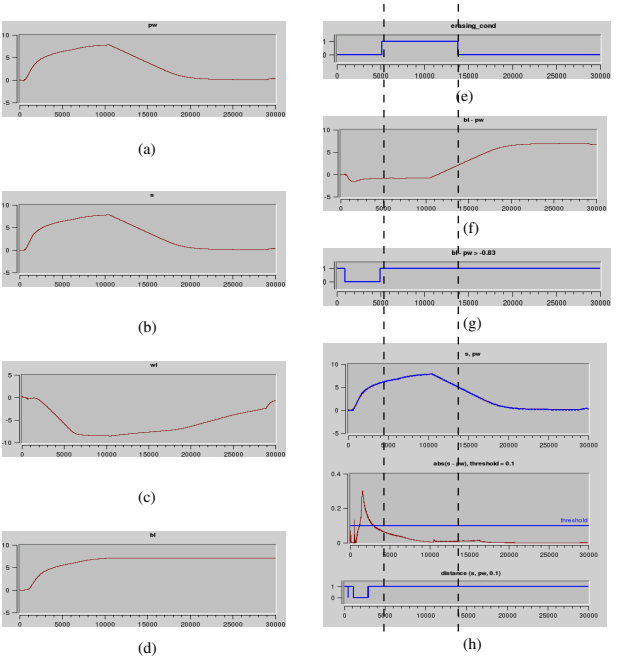


Fig. 4. Erasing Property: (a) pw; (b) s; (c) wl; (d) bl; (e) erasing_cond; (f) bl-pw; (g) bl-pw>=-0.83; (h) distance(s,pw,0.1)

Table 1. Input Size

name	pgm sim # intervals	erase sim # intervals
wl	34829	283624
pw	25478	283037
s	33433	282507
bl	32471	139511
id	375	n/a

Table 2. Offline Algorithm Evaluation

property	time (s)	# intervals
programming1	0.14	99715
programming2	0.42	405907
p-well	0.12	89071
decay	0.50	594709
erasing	2.35	2968578

The execution times of the incremental algorithm are less meaningful because the procedure works in parallel with the simulator which, in most cases, is much more computationally demanding. In fact, one major attraction of the incremental procedure is the ability to detect property violation in the middle of the simulation and save simulation time. Another advantage of the incremental algorithm is its reduced space requirement as we can discard parts of the simulation after they have been fully used. Table 3 compares the memory consumptions of the offline and incremental procedures. For the former we take the total number of intervals generated by the tool while for the latter we take the maximal number of intervals kept simultaneously in memory. We can see that

Table 3. Offline/Incremental Space Requirement Comparison

Property	Offline t = total # intervals	Incremental m = max # active intervals	m/t * 100
programming1	99715	65700	65.9
programming2	594709	242528	40.8
p-well	89071	8	0.01
decay	594709	279782	47.1

this ratio varies a lot from one property to another, going from 0.01% up to 70%. The general observation is that pointwise operators require less memory in the incremental mode, while properties involving the nesting of untimed temporal properties often fail to discard their inputs until the end of the simulation.

6 Conclusions

The main contribution of this paper is the implementation of the AMT tool that monitors temporal properties of continuous and mixed signals. The specification language for describing desired behaviors of continuous signals supported by the tool is STL/PSL, a subset of PSL, properly extended to express sequential properties of such signals. The monitoring algorithms used by AMT are the offline marking procedure from [MN04] and its incremental extension described in this paper. The tool is integrated with numerical simulators by supporting some standard input formats for continuous simulations and by direct communication between the two using a simple protocol built on top of TCP/IP.

AMT was validated through a FLASH memory case-study. The results show that the tool can be effectively used in both its offline and incremental modes. A number of interesting properties concerning transient behavior of continuous signals were described in STL/PSL. Combinations of operators from the analog and temporal layers allow expressing properties such as ramp detection in an input trace, conditional distance-based comparisons between a reference and an input signal, or a stabilization of an input signal around an arbitrary value. The main class of properties that cannot be expressed in STL/PSL are those dealing with the frequency spectrum of signals. A typical English specification of such a property would be "At least 60% of the energy power spectrum of a signal is within its frequency band between 300 and 1500Hz". We hope to introduce such properties into future versions of the tool.

Acknowledgments. We would like to thank Andrea Fedeli, Pierluigi Daglio and Davide Lena from ST Microelectronics Italy, for providing us with the simulations of the FLASH memory cell and their precious help in formulating the properties. We would also like to thank Qiang Lu from Synopsis for providing us with a description of the Nanosim output file format.

References

[ABG+00] Abarbanel, Y., Beer, I., Glushovsky, L., Keidar, S., Wolfsthal, Y.: FoCs: Automatic Generation of Simulation Checkers from Formal Specifications. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 538–542. Springer, Heidelberg (2000)

[ACM02] Asarin, E., Caspi, P., Maler, O.: Timed Regular Expressions. The Journal of the ACM 49, 172–206 (2002)

[AD94] Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)

[ADF+06] Asarin, E., Dang, T., Frehse, G., Girard, A., Le Guernic, C., Maler, O.: Recent Progress in Continuous and Hybrid Reachability Analysis. In: CACSD (2006)

[AFH96] Alur, R., Feder, T., Henzinger, T.A.: The Benefits of Relaxing Punctuality. Journal of the ACM 43, 116–146 (1996) (first published in PODC'91)

[Dru00] Drusinsky, D.: The Temporal Rover and the ATG Rover. In: Havelund, K., Penix, J., Visser, W. (eds.) SPIN Model Checking and Software Verification. LNCS, vol. 1885, pp. 323–330. Springer, Heidelberg (2000)

[DSS+05] D'Angelo, B., Sankaranarayanan, S., Sanchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: Runtime Monitoring of Synchronous Systems. In: TIME'05, pp. 166–174 (2005)

[FGP06] Fainekos, G., Girard, A., Pappas, G.: Temporal Logic Verification Using Simulation. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 171–186. Springer, Heidelberg (2006)

[GO01] Gastin, P., Oddoux, D.: Fast LTL to Büchi Automata Translation. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 53–65. Springer, Heidelberg (2001)

[GPVW95] Gerth, R., Peled, D.A., Vardi, M.Y., Wolper, P.: Simple On-the-fly Automatic Verification of Linear Temporal Logic. In: PSTV, pp. 3–18 (1995)

[HFE04] Havlicek, J., Fisman, D., Eisner, C.: Basic results on the semantics of Accellera PSL 1.1 foundation language, Technical Report 2004.02, Accelera (2004)

[HR01] Havelund, K., Rosu, G.: Java PathExplorer - a Runtime Verification Tool. In: Proc. ISAIRAS'01 (2001)

[KC06a] Konsentini, C., Caspi, P.: Sampling and Voting in Hybrid Computing Systems. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, Springer, Heidelberg (2006)

[KLS+02] Kim, M., Lee, I., Sammapun, U., Shin, J., Sokolsky, O.: Monitoring, Checking, and Steering of Real-time Systems. In: Proc. RV'02. ENTCS vol. 70(4) (2002)

[Koy90] Koymans, R.: Specifying Real-time Properties with Metric Temporal Logic. Real-time Systems 2, 255–299 (1990)

[MMP92] Maler, O., Manna, Z., Pnueli, A.: From Timed to Hybrid Systems. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) Real-Time: Theory in Practice. LNCS, vol. 600, pp. 447–484. Springer, Heidelberg (1992)

[MN04] Maler, O., Nickovic, D.: Monitoring Temporal Properties of Continuous Signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004)

[NMF+06] Nickovic, D., Maler, O., Fedeli, A., Daglio, P., Lena, D.: Analog Case Study, PROSYD Deliverable D3.4/2 (2006), http://www.prosyd.org/twiki/pub/Public/DeliverablePageWP3/prosyd3.4_2.pdf

[MP95] Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, Heidelberg (1995)

[SB00] Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000)

[VW86] Vardi, M.Y., Wolper, P.: An Automata-theoretic Approach to Automatic Program Verification. In: LICS'86, pp. 322–331 (1986)

[Y97] Yovine, S.K.: A Verification Tool for Real-time Systems. International Journal of Software Tools for Technology Transfer 1, 123–133 (1997)

Region Stability Proofs for Hybrid Systems*

Andreas Podelski[1] and Silke Wagner[2]

[1] Universität Freiburg, Germany
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. We present a method and tool (and implementation) for automatic proofs of region stability for hybrid systems. The formal basis of our approach is the new notion of *snapshot sequences*. We use snapshot sequences for a characterization of region stability. Our abstraction-based algorithm checks the conditions in this characterization. A number of experiments demonstrate the practical potential of our approach.

1 Introduction

The problem that we consider in this paper is to show that every trajectory of a given hybrid system stabilizes w.r.t. a given region. This means that every trajectory eventually comes to a point where it lies within the region and never goes out again. Before this point the trajectory can run either inside or outside of the region; i.e., it can reach and leave the region any number of times.

Verification of stability has been so far out of reach for automatic methods. In this paper we propose a new method and tool for the automatic proof of region stability of hybrid systems, to our knowledge for the first time. We have implemented the tool and applied it to a number of benchmarks. Our experiments include the fully automatic stability proof for the break curve behavior of a train system (a previously unsolved benchmark of the AVACS project, `www.avacs.org`).

Our method and tool is possible thanks to a new characterization of region stability. The characterization exploits the fact that region stability is equivalent to the finiteness (NOT boundedness!) of the time that a trajectory can spend outside of the given region. We characterize this finiteness of time in terms of the finiteness of certain state sequences, sequences that we call *snapshot sequences*.

We distinguish three specific kinds of snapshot sequences. For each kind, we compute the constraint that implicitly represents the corresponding set of sequences. This computation is based on a novel source-to-source transformation for hybrid systems. The reachability relation of the transformed hybrid system is the binary reachability relation between snapshots of the original system. We compute an overapproximation of the unary reachability relation of the transformed hybrid system. Here we can use existing methods for reachability analysis for hybrid systems, e.g. [12]. Our approach leverages the abstraction techniques of these methods. Our approach scales with the performance of these abstraction techniques. We present an implementation of our method

* 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). See `www.avacs.org` for more information.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 320–335, 2007.

and tool and use it to demonstrate the practical potential of our method on several benchmarks.

2 Preliminaries

We follow the standard terminology and notation for hybrid systems, states and trajectories.

A **hybrid system** is a tuple (fixed from now on)

$$A = (\mathcal{L}, \mathcal{V}, (\mathsf{jump}_{\ell,\ell'})_{\ell,\ell' \in \mathcal{L}}, (\mathsf{flow}_\ell)_{\ell \in \mathcal{L}}, (\mathsf{inv}_\ell)_{\ell \in \mathcal{L}}, (\mathsf{init}_\ell)_{\ell \in \mathcal{L}})$$

consisting of the following components:

1. a finite set $\mathcal{L}$ of locations.
2. a finite set $\mathcal{V}$ of real-valued variables.
3. a family $(\mathsf{jump}_{\ell,\ell'})_{\ell,\ell' \in \mathcal{L}}$ of formulas over $\mathcal{V}$ and $\mathcal{V}''$ representing the possible jumps from location ℓ to location ℓ'.
4. a family $(\mathsf{flow}_\ell)_{\ell \in \mathcal{L}}$ of formulas over $\mathcal{V}$ and $\dot{\mathcal{V}}$ specifying the continuous variable update in location ℓ. We use $\dot{\mathcal{V}} = \{\dot{x}_1, \dot{x}_2, \ldots\}$ for the set of dotted variables. A variable $\dot{x}$ represents the first derivative of x with respect to time, i.e. $\dot{x} = dx/dt$.
5. a family $(\mathsf{inv}_\ell)_{\ell \in \mathcal{L}}$ of formulas over $\mathcal{V}$ representing the invariant condition in location ℓ.
6. a family $(\mathsf{init}_\ell)_{\ell \in \mathcal{L}}$ of formulas over $\mathcal{V}$ representing the initial states of the system.

A **state** s is a pair (ℓ, ν) consisting of a location ℓ of $\mathcal{L}$ and a valuation ν of all variables over the set $\mathcal{V}$. A set of states is also called a **region**. In the remainder of this paper we restrict ourselves to **interval regions**, i.e. to regions φ that are given by $\varphi \equiv x \in [x_{min}, x_{max}]$, where x is a continuous variable of the hybrid system and x_{min} and x_{max} are constants (including $\pm\infty$) with $x_{min} < x_{max}$.

A **trajectory** τ of a hybrid system A is a function mapping time points t in $\mathbb{R}^+$ to states in Σ such that the following conditions hold:

1. If $\tau(0)$ has location ℓ, then $\tau(0)$ must satisfy the initial condition of that location.

$$\tau(0) \models \mathsf{init}_\ell$$

2. If the real-valued component ν of τ is differentiable at t, and both $\tau(t)$ and the left-limit of τ at t, $\lim_{t' \to t_-} \tau(t')$, have the same location ℓ, then the pair $(\nu, \dot{\nu})$ of variable valuation and valuation of the first derivatives satisfies the invariant and the flow condition of location ℓ.

$$(\nu, \dot{\nu}) \models \mathsf{inv}_\ell \wedge \mathsf{flow}_\ell$$

3. If the left-limit of τ at t has location ℓ and $\tau(t)$ has a different location ℓ', then the real-valued component of the left-limit of τ at t and $\tau(t)$ must satisfy the jump condition from location ℓ to location ℓ'.

$$(\lim_{t' \to t_-} \tau(t'), \tau(t)) \models \mathsf{jump}_{\ell,\ell'}$$

Definition 1 (Region stability [19]). *A hybrid system is* stable *with respect to a region* φ *if for every trajectory* τ *there exists a point of time* t_0 *such that from then on, the trajectory is always in the region* φ.

$$\forall \tau \, \exists t_0 \, \forall t \geq t_0 \, : \, \tau(t) \in \varphi$$

Region stability is essentially what is called *practical stability* in [16].

3 Examples

We next present our benchmarks which are typical instances of the verification problem. The formal description of the hybrid systems can be found in [20]. Each of the next eight benchmarks is to illustrate a particular aspect of the verification problem. The other three benchmarks describe three well-known scenarios.

Example 1. Our first example is a hybrid system with one location and one continuous variable x. Initially the value of x is greater than 0; the flow condition is given by $\dot{x} = -1$. The region with respect to which we want to prove stability is given by $x \leq 0$. Intuitively it is clear that all trajectories of the system must end up in the region φ (since the value of x is strictly monotonically decreasing by -1). For every single trajectory the amount of time that the trajectory can spend outside of the region φ is finite. However, the time that a trajectory can spend outside of φ is unbounded.

Example 2: A simple heating system. In our second example we consider a well-known heating system for a room. Either the heater is off and the temperature x falls or the heater is on and the temperature rises. This system is not stable in the classical sense (w.r.t. an equilibrium point). We want to show stability w.r.t. the region $x \in [65, 82]$.

Example 3: A more complex heating system. Our next example is a modification of the heating system that we have seen in the last example. The modified heating system consists of three continuous variables x_p, x_e and t; x_p stands for the temperature of the room and x_e for the temperature of an internal engine. The internal engine may overheat and switch off the heater temporarily, even though the desired temperature for the room (given by $x_p \in [65, 82]$) is not yet reached. This means that, starting from low, the temperature will not increase strictly monotonically but it will also decrease during some periods of time. A side effect of this behavior is that a trajectory can reach the desired region but leaves it again for some time before it stabilizes.

Example 4: A bouncing ball. This example is a modification of the well-known bouncing ball. A ball (thought of as a point-mass) is thrown horizontally against a wall. The distance between the wall and the thrower is denoted by x_t, the distance between the wall and the ball is denoted by x_b. We assume that the ball has a constant speed and does not lose any energy with a bounce. As soon as the thrower has thrown the ball he moves towards the wall until the ball returns to him; then he throws the ball again.

Each execution of the hybrid system that models this scenario is a Zeno execution, this means a solution of the system having infinitely many discrete jumps in finite time. Nevertheless we can show that the thrower can come arbitrarily close to the wall, i.e. we can prove stability of the system w.r.t. the region $x_t \leq \varepsilon$ for every $\varepsilon > 0$.

Example 5: An one-tank water system. In the following example we consider a one-tank water system with a constant inflow of water. The volume of water in the tank is denoted by x. The tank has a pipe such that water can also flow out of the tank again. The pipe can be opened for at most 8 seconds; after that the pipe must be closed again for 10 seconds. We want to know whether the tank can be drained, no matter what the initial volume of water is; i.e. we must check whether the system is stable w.r.t. $x <= 0$.

As in Example 1, the time that a trajectory of this system can spend outside of the desired region is unbounded. Furthermore we prove in this example stability w.r.t. the equilibrium point $x = 0$. (Since the invariants of the system assure that $x \geq 0$, stability w.r.t. $x \leq 0$ implies stability w.r.t. the equilibrium point $x = 0$.)

Example 6: A distance controller. The next example is a model of a distance controller. We consider two cars driving one after another. The leading car has a constant speed $v_1 = 50$. The second car is governed by a controller that continuously senses the distance x between the two cars. If the distance is greater than a given value the second car speeds up; if the distance is smaller than a second value it slows down. The second car has a maximum speed of 70 and a minimum speed of 0. The goal is to prove that the distance x between the two cars is always > 0.

Example 7: A two-tank water system. Now we consider a two-tank water system consisting of two tanks one upon the other. The variables x_1 and x_2 denote the volumes of water in the upper tank 1 and the lower tank 2. Water flows constantly out of the system from the lower tank. The system can switch on or off the inflow of water into the upper tank, and the flow of water from the upper to the lower tank; but both tanks must not overflow. The objective is to keep the water volume of the lower tank above 6, i.e. we are interested in stability w.r.t. the region $x_2 > 6$.

Example 8: Train brakes. In our next example we consider the braking behavior of a train, see Fig. 1. Initially the train is moving with a constant speed v. Eventually it starts braking, either with one break (if the speed is ≤ 200) or with two brakes (if the speed is > 200). There is a time delay between ordering the brake application and reaching the full brake effort. The braking capacity of the train depends on the speed. At a speed of 200 the second brake is released again. We want to prove stability of the system w.r.t. $v \leq 0$, i.e. we want to show that the train can always stop.

Three more examples. Each of the previous benchmarks illuminates a particular facet of verifying region stability. The following benchmarks do not illustrate a new angle on stability but they describe three well-known (realistic) scenarios.

1. RLC circuit: we analyze a series RLC circuit consisting of one resistor, one inductor and one capacitor.
2. Damped oscillation of a pendulum: we show that the amplitude of the oscillation will always go towards 0.
3. The energy of an exothermic chemical reaction: we prove that the energy of an exothermic chemical process tends to 0.

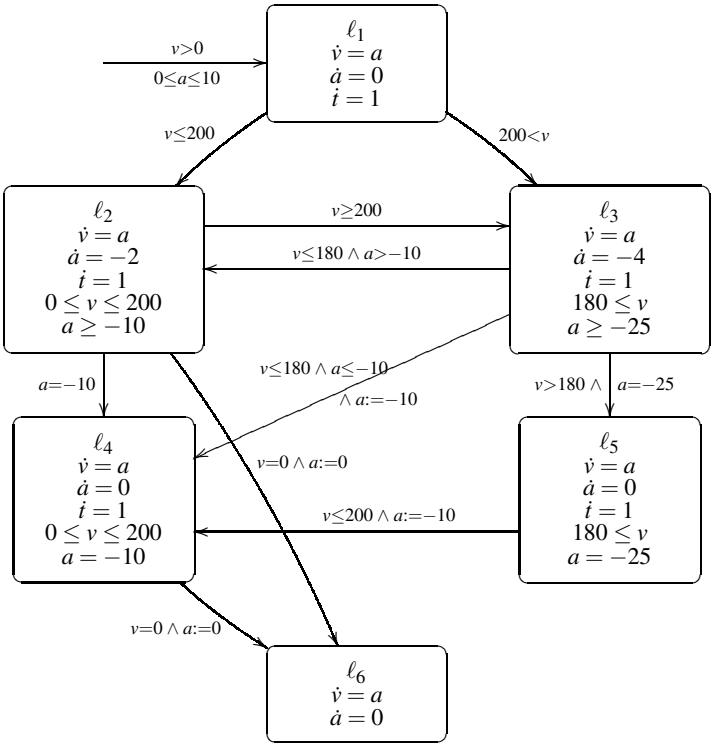


Fig. 1. Train brakes

Experimental Results. We have implemented the algorithm described in the next sections (5.1 and 5.2) and applied it to the examples listed above, see Fig. 2. To our knowledge these experiments are the first automatic proofs for the stability of those hybrid systems.

4 Snapshot Sequences

In this section we give a characterization of region stability that serves as a basis for our algorithm. For reasons of space we have to omit a lot of formal details. A technical report [20] allows a sceptical reader to access the correctness of our results.

Our characterization exploits the fact that region stability is equivalent to the finiteness (NOT boundedness) of the time that a trajectory can spend outside of the given region. We will characterize this finiteness of time in terms of the finiteness of certain sequences of states.

Definition 2 (Snapshot sequence.). *Given a hybrid system A and a region φ, a* snapshot sequence $s_0, s_1, s_2, \ldots$ *is a sequence of states such that (i) all states of the sequence lie on the same trajectory τ of A, (ii) all states are not in the region φ, and (iii) all pairs*

System	# Variables	# Locations	Run Time
Example 1	2	1	0.191s
Simple heater	3	2	0.490s
Complex heater	3	2	1.920s
Bouncing ball	3	2	4.209s
One tank system	3	3	1.813s
Distance controller	3	4	1.186s
Two tank system	3	4	16.545s
Train brake	3	6	2.589s
RLC circuit	2	2	0.449s
Pendulum	2	2	0.264s
Exothermic reaction	2	3	0.428s

Fig. 2. Experimental results (on a Pentium M 1,7GHz processor running Debian Linux 2.6.7) yielding fully automatic stability proofs of the hybrid systems

of consecutive states have a minimum time distance δ*, where* δ *is an arbitrary but fixed constant greater than 0.*

(i) $\exists \tau \, \forall i \, \exists t_i \; : \; s_i = \tau(t_i)$,
(ii) $\forall i \; : \; s_i \notin \varphi$,
(iii) $\exists \delta > 0 \, \forall i \; : \; t_{i+1} - t_i \geq \delta$.

We now define the three kinds of snapshot sequences. In Section 5 we explain how one can compute an effective representation of the set of snapshot sequences for each of the three kinds: 1. snapshot sequences on monotonic flows, 2. snapshot sequences of extremal-points, and 3. snapshot sequences of entry points.

1. Snapshot sequences on monotonic flows. To illustrate the first kind of snapshot sequences we consider a (strictly) monotonic trajectory of a linear hybrid systems with one location and one continuous variable x.

A monotonic trajectory that is stable can never leave the region φ again after it has reached it *once*. Or the other way round: a monotonic trajectory that is not stable either never reaches the region or reaches the region but leaves it again for good. In both cases the amount of time that the (unstable) trajectory spends outside of the given region is infinite. This means if we consider an arbitrary time-divergent discretization (by a constant time step $\delta > 0$) of a monotonic trajectory then only finitely many states of the discretization are outside the region φ if and only if the trajectory is stable.

Snapshot sequences arising by equidistant discretizations of a monotonic trajectory without jumps are the first kind of snapshot sequences we are interested in.

2. Sequences of extremal-points. For non-monotonic trajectories it is not the case that the finiteness of states of any arbitrary discretization outside of the region φ is equivalent to the stability of the trajectory. If we consider for example the trajectory in Fig. 3 and we disretize it by 2π then all states of the discretization are in the region φ although the trajectory is not stable.

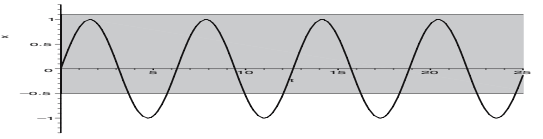


Fig. 3. Sample trajectory of a hybrid system with one location, one continuous variable x and flow $\dot{x}(t) = \sin(t)$. The trajectory violates stability w.r.t. the grey region $\varphi \equiv x \in [-0.5, 1.1]$.

It does also not suffice to consider the equidistant discretizations of each monotonic part of the trajectory separately: in the example of Fig. 3 all of them are finite outside of the region φ (independent of the discretization width δ); but the trajectory consists of *infinitely many* monotonic parts.

A trajectory changes its monotonicity behavior (i.e. its direction) during a continuous flow at the so-called *extremal-points*. We make the following observation for hybrid systems with one location: if a trajectory has only finitely many extremal-points outside of the region φ (this means the trajectory consists only of finitely many monotonic parts outside of φ) *and* if all monotonic parts are finite outside of φ then the trajectory is stable w.r.t. φ.

Sequences of extremal-points are the second kind of snapshot sequences we are interested in.

3. Sequences of entry-points. Now we consider a (non-stable) trajectory of a a hybrid system with two locations and one variable x, see Fig 4. In one location the value of x is increasing, in the other location the value of x is decreasing.

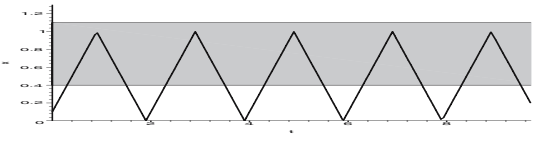


Fig. 4. A sample trajectory of a hybrid system with two locations. The system is not stable w.r.t. the grey region $\varphi \equiv x \in [0.4, 1]$.

This scenario and the last one (Fig. 3) are quite similar. For the same reasons as we have seen there we must again assure that the trajectory consists of only finitely many monotonic parts outside of the region φ in order to prove stability. The trajectory in Fig. 4 changes its monotonicity behavior during discrete jumps. We call states just after a discrete jump *entry-points*.

Sequences of entry-points are the third kind of snapshot sequences we are interested in.

We can now formulate the three conditions that together characterize region stability of hybrid systems.

Condition 1 There is no infinite snapshot sequence such that
(i) no entry-point lies between two states of the sequence and
(ii) no extremal-point lies between two states of the sequence.

Condition 2 There is no infinite snapshot sequence such that
all states of the sequence are extremal-points.

Condition 3 There is no infinite snapshot sequence such that
all states of the sequence are entry-points.

Theorem 1. *Condition 1, Condition 2, and Condition 3 together are sufficient and necessary for region stability of a hybrid system A w.r.t. an interval region* φ.

Proof. (Sketch.)

if-direction: We assume a hybrid system A and an interval region $\varphi \equiv x \in [x_{min}, x_{max}]$. Furthermore we assume that the Conditions 1, 2, and 3 hold, but A has a non-stabilizing trajectory τ.
By Condition 3 there must be a time point t_1 such that from then on no more entry-point lies on τ outside of φ. Similarly, by Condition 2 there is a time point t_2 such that from t_2 on no more extremal-point w.r.t. x lies on τ outside of φ. W.l.o.g. we assume $t_2 > t_1$.
Either all states after t_2 lie outside of φ. Or there is a state on τ after t_2 that lies inside of the region φ. Since τ is a non-stabilizing trajectory it will leave φ again afterwards, say at t_3. No more entry-point and no more extremal-point will occur outside of φ, hence the trajectory τ will move away from the region φ forever after t_3.
In other words, after t_3 the computation of the system proceeds in one location, say ℓ, and the flow is monotonic. But this means that the sequence

$$\tau(t_3),\ \tau(t_3+\delta),\ \tau(t_3+2\delta),\ \ldots$$

(for any arbitrary $\delta > 0$) is an infinite snapshot sequence on the same flow without entry- and extremal-points in between, a contradiction to Condition 1.

only if-direction: The only if-direction follows from the fact that "stability with respect to φ" implies that all possible snapshot sequences are finite. □

We can extend our result to n-dimensional regions, i.e. to regions φ that can be expressed as cartesian products of intervals.

$$\varphi \quad \equiv \quad (x_1,\ldots,x_n) \ \in \ [x_1^{min}, x_1^{max}] \times \ldots \times [x_n^{min}, x_n^{max}] \,,$$

We call such regions **box regions**. The idea is to prove that the hybrid system is stable w.r.t. each single one-dimensional component φ_i of the cartesian product,

$$\varphi_i \quad \equiv \quad x_i \in [x_i^{min}, x_i^{max}] \,, \qquad i = 1 \ldots n \,.$$

The following lemma shows that this yields stability w.r.t. the n-dimensional region φ.

Lemma 1. *A hybrid system A is stable w.r.t. an n-dimensional box region* φ *if and only if A is stable w.r.t. each interval region* φ_i.

Proof. **if-direction:** Assume that A is stable with respect to to each one-dimensional region φ_i, formally

$$\forall i \ \forall \tau \ \exists t_0^i \ \forall t \geq t_0^i \ : \ \tau(t) \in \varphi_i \ .$$

We take the maximum over all time points t_0^i and call it t_0.

$$t_0 \ = \ \max_{i=1\ldots n} t_0^i$$

This means that from the time point t_0 on the trajectory τ is in all one-dimensional regions φ_i and hence in the n-dimensional box region φ, which implies that A is stable with respect to φ.

only if-direction: Assume that A is stable with respect to n-dimensional box region φ, i.e.

$$\forall \tau \ \exists t_0 \ \forall t \geq t_0 \ : \quad \tau(t) \in \varphi \ .$$

Especially this means that from t_0 on the trajectory τ is in all one-dimensional interval regions φ_i.

$$\forall \tau \ \exists t_0 \ \forall t \geq t_0 \ : \ \tau(t) \in \varphi_1 \ \wedge \ \ldots \ \tau(t) \in \varphi_n \ .$$

Thus A is stable with respect to each interval region φ_i. □

5 Algorithm

Fig. 5 shows the control flow graph of the overall algorithm. The input is a hybrid system A and a region φ. The first step of the algorithm is the computation of a transformed hybrid system such that the reachability relation of the transformed hybrid system is the binary reachability relation between snapshots of the original system. The second step is the computation of the reachability relation of the transformed hybrid system. We have implemented this step using PHAVer, a tool for (unary) reachability analysis [12]. The last step of the algorithm is to check whether all relations in the output of PHAVer are well-founded. (*Well-founded* means that there is no infinite sequence of states $s_1, s_2, s_3, \ldots$ such that each pair of consecutive states (s_i, s_{i+1}) satisfies the relation.) Our implementation uses RankFinder for the well-foundedness test [21, 17]. The output of the algorithm is a Yes / Don't know answer.

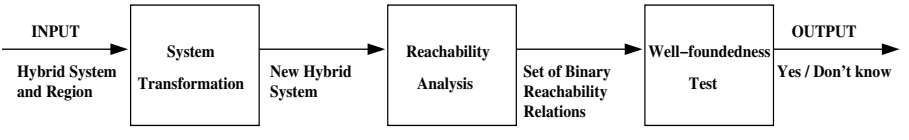


Fig. 5. Control flow graph of the overall algorithm. Our implementation calls PHAVer for the reachability analysis and RankFinder for the well-foundedness test.

5.1 System Transformation

The goal of this section is to show how one can compute an effective representation of the set of snapshot sequences for each of the three kinds (see Section 4). In order to check finiteness of snapshot sequences we give an implicit representation of the set of these sequences by constraints that denote binary relations. (A binary relation represents the set of all sequences such that each pair of consecutive elements lies in the relation).

We distinguish three different kinds of snapshot sequences; for each kind we compute the constraint that implicitly represents the corresponding set of sequences. For each of the three cases the algorithm to compute the constraints is based on the syntactic transformation of the original hybrid system into another one such that the reachability relation of the transformed hybrid system is the binary reachability relation between snapshots of the original system; an overapproximation of the unary reachability relation of the transformed hybrid system is computed by using dedicated abstraction techniques that have been developed for safety proofs of hybrid systems (see e.g. [12]).

We will now use an example to explain how we compute a set of relations that together represent all possible snapshot sequences (possibly more, since we use an overapproximation).

Snapshot sequences on monotonic flows. We describe informally how one can compute the representation for all snapshot sequences on a monotonic flow. The transformation for the computation of the other kinds of snapshot sequences is similar.

We consider a hybrid system with one location ℓ and one continuous variable x (see Fig. 6). The flow condition is given by

$$\dot{x} \;=\; \sin(x)\;.$$

We want to know whether the system is stable w.r.t. the region $\varphi \;\equiv\; x \in [-1,1]$.

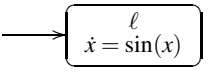


Fig. 6. Hybrid system with one location and one continuous variable

Fig. 7 shows the system transformation for the computation of snapshot sequences on monotonically *increasing* parts of a trajectory. (The transformed system for the *de*creasing parts is quite similar; the only difference is that the guard $\sin(x) > 0$ and the invariant $\sin(x') > 0$ are replaced by $\sin(x) < 0$ and $\sin(x') < 0$.)

The transformed system A^T has two continuous variables, namely the original variable x and its copy x'. Initially the values of x and x' are the same. In the location ℓ^0 the continuous flows of the two variables are identical, each of which corresponding to the flow of the original system.

$$\mathsf{flow}^T_{\ell^0}(x,x',\dot{x},\dot{x}') \quad\equiv\quad \mathsf{flow}_\ell(x,\dot{x}) \;\wedge\; \mathsf{flow}_\ell(x',\dot{x}')$$

This means that one can view a state (s,s') of the transformed system in ℓ^0 as a pair of identical states of the original system.

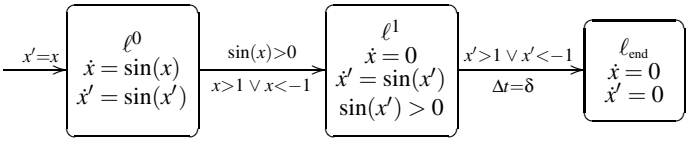


Fig. 7. Transformed system for the computation of snapshot sequences on a monotonic flow

The transformed system can jump from the location ℓ^0 to the location ℓ^1 only if the first derivative of x is > 0 (meaning that the values of x are monotonically increasing) and if the value of x is not in the region φ (i.e. either if $x > 1$ or if $x < -1$).

$$\mathsf{jump}^T_{\ell^0,\ell^1}(x,x') \quad \equiv \quad \sin(x) > 0 \wedge x \notin \varphi$$

Until the discrete jump the values of x and x' are identical. After the jump only x' continues evolving as before whereas x is fixed from now on (in the location ℓ^1 the flow of x is constantly 0). The values of x' are monotonically increasing according to the invariant condition $\sin(x') > 0$ of ℓ^1.

$$\begin{aligned} \mathsf{inv}^T_{\ell^1}(x,x') &\equiv \mathsf{inv}_\ell(x,x') \wedge \sin(x') > 0 \\ \mathsf{flow}^T_{\ell^1}(x,x',\dot{x},\dot{x}') &\equiv \dot{x} = 0 \wedge \mathsf{flow}_\ell(x',\dot{x}') \end{aligned}$$

This means that in ℓ^1 one can view a state (s,s') of the transformed system as a pair of states on a monotonically increasing part of a trajectory of the original system, where the first state s is not in the region φ.

To make sure that the value of x' is also not in φ the transformed system can jump to the third location ℓ_{end} only if $x' > 1$ or if $x' < -1$. Additionally the jump condition ensures that the transformed system must spend time δ in the location ℓ^1.

$$\mathsf{jump}^T_{\ell^1,\ell_{\mathsf{end}}}(x,x') \quad \equiv \quad x' \notin \varphi \wedge \Delta t = \delta$$

In ℓ_{end} the values of both variables x and x' are fixed. A state (s,s') of the transformed system in ℓ_{end} can be viewed as a pair of states on a monotonically increasing part of a trajectory of the original system where both states are not in φ and have a time distance δ. (The constant $\delta > 0$ is the discretization width of the trajectory.)

Altogether this means that the reachability relation of the transformed hybrid system in ℓ_{end} is the binary reachability relation between snapshots of the original system that arise by equidistant discretizations of a monotonically increasing part of a trajectory outside of the region φ.

Formal Description of the System Transformation Given a hybrid system

$$A = \big(\mathcal{L}, \mathcal{V}, (\mathsf{flow}_\ell), (\mathsf{jump}_{\ell,\ell'}), (\mathsf{inv}_\ell), (\mathsf{init}_\ell)\big)$$

for which we want to prove stability w.r.t. the interval region

$$\varphi \quad \equiv \quad x \in [x_{min}, x_{max}]\,;$$

the transformed system A^T is given by:

1. Locations: Each location ℓ_i of the original system corresponds to five locations $\ell_i^0,\ldots,\ell_i^4$ in the transformed system. We refer to the set of all locations from ℓ_1^k to ℓ_m^k as $\mathcal{L}^k$,
$$\mathcal{L}^k = \{\ell_1^k,\ldots,\ell_m^k\}\ ,\ k \in \{0,1,2,3,4\}\ .$$
In addition, the transformed system has two locations ℓ_{init} and ℓ_{end}. Altogether, the set $\mathcal{L}^T$ of locations of the transformed system consists of the following components:
$$\mathcal{L}^T \ = \ \{\ell_{\mathsf{init}}\} \cup \mathcal{L}^0 \cup \mathcal{L}^1 \cup \mathcal{L}^2 \cup \mathcal{L}^3 \cup \mathcal{L}^4 \cup \{\ell_{\mathsf{end}}\}$$

2. Variables: The set $\mathcal{V}^T$ of variables of the transformed system contains all variables of $\mathcal{V}$, their primed versions, and an additional variable called flag.
$$\mathcal{V}^T \ = \ \mathcal{V} \cup \{\mathsf{flag}\} \cup \mathcal{V}'$$

3. Flow constraints: First, in the locations ℓ_{init} and ℓ_{end} the flow of all continuous variables is 0.
$$\begin{aligned}\mathsf{flow}^T_{\ell_{\mathsf{init}}}(x_1,\ldots,t',\dot{x}_1,\ldots,\dot{t}') &\equiv \bigwedge_{x\in\mathcal{V}^T} \dot{x}=0\\ \mathsf{flow}^T_{\ell_{\mathsf{end}}}(x_1,\ldots,t',\dot{x}_1,\ldots,\dot{t}') &\equiv \bigwedge_{x\in\mathcal{V}^T} \dot{x}=0\end{aligned}$$
In each location ℓ_i^0 of $\mathcal{L}^0$, the flow of the variables $x_1,\ldots,t$ in the transformed system is the same as the flow of $x_1,\ldots,t$ in the original system; each variable $x_1',\ldots,t'$ behaves exactly like its unprimed version, that is the flow of $x_1',\ldots,t'$ is equal to the flow of the original system after replacing the variables $x_1,\ldots,t$ by their primed versions $x_1',\ldots,t'$.
$$\begin{aligned}\mathsf{flow}^T_{\ell_i^0}(x_1,\ldots,t',\dot{x}_1,\ldots,\dot{t}') \equiv\ & \mathsf{flow}_{\ell_i}(x_1,\ldots,\dot{t}) \wedge \dot{\mathsf{flag}}=0\\ & \wedge \mathsf{flow}_{\ell_i}(x_1',\ldots,\dot{t}')\end{aligned}$$
In each location of $\mathcal{L}^1\cup\ldots\cup\mathcal{L}^4$ the values of the variables $x_1,\ldots,t$ are fixed, i.e. the flow of them is constantly 0. The variables $x_1',\ldots,t'$ keep on evolving as before.
$$\mathsf{flow}^T_{\ell_i^k}(x_1,\ldots,t',\dot{x}_1,\ldots,\dot{t}') \equiv \Big(\bigwedge_{x\in\mathcal{V}\cup\{\mathsf{flag}\}} \dot{x}=0\Big) \wedge \mathsf{flow}_{\ell_i}(x_1',\ldots,\dot{t}')\ ,\quad k\in\{1,\ldots,4\}$$

4. Jump constraints: A jump is possible from the location ℓ_{init} to a location ℓ_i^0 of $\mathcal{L}^0$ if the initial condition of the location ℓ_i of the original system A is fulfilled for the variables $(x_1,\ldots,t)$.
$$\mathsf{jump}^T_{\ell_{\mathsf{init}},\ell_i^0}(x_1,\ldots,t') \equiv \mathsf{init}_{\ell_i}(x_1,\ldots,t)$$
The second kind of jumps are jumps between two locations of $\mathcal{L}^0$ and between two locations of $\mathcal{L}^4$, respectively. The condition for a jump between the location

ℓ_i^0 and the location ℓ_j^0 for the variables $(x_1,\ldots,t,x_1',\ldots,t')$ corresponds to the jump condition between the locations ℓ_i and ℓ_j of the original system A for the variables $(x_1,\ldots,t)$. Similarly a jump condition between the locations ℓ_i^4 and ℓ_j^4 of the transformed system corresponds to the jump condition from location ℓ_i to ℓ_j of the original system after replacing the variables $x_1,\ldots,t$ by their primed versions.

$$\begin{aligned}\mathsf{jump}^T_{\ell_i^0,\ell_j^0}(x_1,\ldots,t') &\equiv \mathsf{jump}_{\ell_i,\ell_j}(x_1,\ldots,t)\\ \mathsf{jump}^T_{\ell_i^4,\ell_j^4}(x_1,\ldots,t') &\equiv \mathsf{jump}_{\ell_i,\ell_j}(x_1',\ldots,t')\end{aligned}$$

To compute sequences on monotonic flows we allow jumps from $\ell_i^0 \in \mathcal{L}^0$ to $\ell_i^1 \in \mathcal{L}^1$ (and to $\ell_i^2 \in \mathcal{L}^2$, respectively) if the first derivative of x is greater than 0 (and less than 0, respectively) and if x does not lie in φ.

$$\begin{aligned}\mathsf{jump}^T_{\ell_i^0,\ell_i^1}(x_1,\ldots,t') &\equiv \dot{x} > 0 \wedge \neg\varphi\\ \mathsf{jump}^T_{\ell_i^0,\ell_i^2}(x_1,\ldots,t') &\equiv \dot{x} < 0 \wedge \neg\varphi\end{aligned}$$

Similarly the transformed system can take a jump from a location $\ell_i^0 \in \mathcal{L}^0$ to a location $\ell_i^3 \in \mathcal{L}^3$ if the first derivative of x is 0 and if x does not lie in φ.

$$\mathsf{jump}^T_{\ell_i^0,\ell_i^3}(x_1,\ldots,t') \equiv \dot{x} = 0 \wedge \neg\varphi$$

A jump from ℓ_i^1 to ℓ_{end} (and from ℓ_i^2 to ℓ_{end}, respectively) is possible if the first derivative of x' is greater than 0 (and less than 0, respectively), if x' does not lie in φ, and if the system has spend time δ in the location ℓ_i^1 (and in ℓ_i^2, respectively).

$$\begin{aligned}\mathsf{jump}^T_{\ell_i^1,\ell_{\mathsf{end}}}(x_1,\ldots,t') &\equiv \dot{x}' > 0 \wedge \neg\varphi \wedge t'-t = \delta\\ \mathsf{jump}^T_{\ell_i^2,\ell_{\mathsf{end}}}(x_1,\ldots,t') &\equiv \dot{x}' < 0 \wedge \neg\varphi \wedge t'-t = \delta\end{aligned}$$

Similarly the transformed system can jump from from ℓ_i^3 to ℓ_{end} if the system has spend time δ in the location ℓ_i^3, if the first derivative of x' is 0, and if x' is not in φ.

$$\mathsf{jump}^T_{\ell_i^3,\ell_{\mathsf{end}}}(x_1,\ldots,t') \equiv \dot{x}' = 0 \wedge \neg\varphi \wedge t'-t \geq \delta$$

For the computation of pairs of entry-points of the original system we need a jump from a location $\ell_i^0 \in \mathcal{L}^0$ to a location $\ell_j^4 \in \mathcal{L}^4$ whenever the jump condition between the locations ℓ_i and ℓ_j of the original system A is possible but only if x is not in the region φ. During the jump the value of flag is set to the index j of the target location.

$$\mathsf{jump}^T_{\ell_i^0,\ell_j^4}(x_1,\ldots,t') \equiv \mathsf{jump}_{\ell_i,\ell_j}(x_1,\ldots,t) \wedge \neg\varphi \wedge \mathsf{flag} := j$$

The transformed system can jump from $\ell_i^4 \in \mathcal{L}^4$ to ℓ_{end} if x is not in φ, the value of flag is j, and the jump condition from ℓ_i to ℓ_j of the original system A holds for the primed variables. Additionally we must guarantee the discretization width δ (for $\delta > 0$ constant).

$$\begin{aligned}\mathsf{jump}^T_{\ell_i^4,\ell_{\mathsf{end}}}(x_1,\ldots,t') \equiv \bigvee_{\ell_j\in\mathcal{L}} \Big(&\mathsf{jump}_{\ell_i,\ell_j}(x_1',\ldots,t') \wedge \neg\varphi\\ &\wedge \mathsf{flag} = j \wedge t'-t \geq \delta\Big)\end{aligned}$$

5. Invariant conditions: For the locations ℓ_{init} and ℓ_{end} the invariant condition is true.

$$\begin{aligned} \mathsf{inv}^T_{\ell_{\mathsf{init}}}(x_1,\ldots,t') &\equiv \text{true} \\ \mathsf{inv}^T_{\ell_{\mathsf{end}}}(x_1,\ldots,t') &\equiv \text{true} \end{aligned}$$

For a location ℓ_i^0 in $\mathcal{L}^0$ the invariant condition over $x_1,\ldots,t,x_1',\ldots,t'$ is the same as the invariant condition of the original system A for ℓ_i over $x_1,\ldots,t$.

$$\mathsf{inv}^T_{\ell_i^0}(x_1,\ldots,t') \equiv \mathsf{inv}_{\ell_i}(x_1,\ldots,t)$$

For a location $\ell_i^1 \in \mathcal{L}^1$ (and $\ell_i^2 \in \mathcal{L}^2$, respectively) the invariant condition over $x_1,\ldots,t,x_1',\ldots,t'$ corresponds to the invariant condition of the original system A for ℓ_i over $x_1',\ldots,t'$ in addition to the condition $\dot{x} > 0$ (and $\dot{x} < 0$, respectively).

$$\begin{aligned} \mathsf{inv}^T_{\ell^1}(x_1,\ldots,t') &\equiv \mathsf{inv}_{\ell}(x_1',\ldots,t') \wedge \dot{x} > 0 \\ \mathsf{inv}^T_{\ell^2}(x_1,\ldots,t') &\equiv \mathsf{inv}_{\ell}(x_1',\ldots,t') \wedge \dot{x} < 0 \end{aligned}$$

For a location $\ell_i^3 \in \mathcal{L}^3$ or $\ell_i^4 \in \mathcal{L}^4$ the invariant condition over $x_1,\ldots,t,x_1',\ldots,t'$ is the same as the invariant condition of the original system A for ℓ_i over $x_1',\ldots,t'$

$$\begin{aligned} \mathsf{inv}^T_{\ell_i^3}(x_1,\ldots,t') &\equiv \mathsf{inv}_{\ell_i}(x_1',\ldots,t') \\ \mathsf{inv}^T_{\ell_i^4}(x_1,\ldots,t') &\equiv \mathsf{inv}_{\ell_i}(x_1',\ldots,t') \end{aligned}$$

6. Initial conditions: Initially, each variable x_i has the same value as x_i', the value of t is equal to the value of t', and the value of flag is set to 0; the system starts at time point $t = 0$ in the location ℓ_{init}.

$$\begin{aligned} \mathsf{init}^T_{\ell_{\mathsf{init}}} &\equiv \bigwedge_{x \in \mathcal{V}} x = x' \wedge t = 0 \wedge \mathsf{flag} = 0 \\ \mathsf{init}^T_{\ell} &\equiv \text{false} \quad \forall\, \ell \neq \ell_{\mathsf{init}} \end{aligned}$$

5.2 Reachability Analysis and Well-Foundedness Tests

Now we are given a transformed hybrid system such that the reachability relation of the transformed hybrid system is the binary reachability relation between snapshots of the original system. Our implementation uses PHAVer [12] for the reachability analysis of the transformed system. The output of PHAVer is a set of constraints, given by a disjunction of conjunctions of linear inequalities. The constraints denote binary relations of the original hybrid system.

We check for each single relation that it is well-founded. *Well-founded* means that there is no infinite sequence of states $s_1, s_2, s_3, \ldots$ such that each pair of consecutive states (s_i, s_{i+1}) satisfies the relation. For the well-foundedness tests our implementation uses RankFinder [21,17]. It is indeed sufficient to prove well-foundedness for each relation separately to show that all snapshot sequences (for each kind) are finite outside of the region φ, which means that the original hybrid system is stable w.r.t. the region φ [18].

We have implemented the algorithm described above and applied it on the benchmarks listed in Section 3. The results can be found in Fig. 2.

6 Related Work

There are many proof rules for a wide variety of notions of stability. They are all based on Lyapunov theory [2,3,14,15,16]. The automation of these proof rules works by the synthesis of Lyapunov functions, usually by a range of sophisticated arithmetic constraint solving methods, see e.g. [4,5].

Existing tools for verifying region stability fall into two classes: (1) tools that abstract hybrid systems into finite state models and apply model checkers to this abstraction [11]; and (2) tools that use unary reachability to check that after a given time point t all states outside of the stable region are unreachable [13]. The first class suffers from the fact that abstraction to finite state models is intrinsically inappropriate for systems with finite but unboundedly long trajectories outside of the region (in a finite state system where all executions have finite length, the executions must have bounded length); the second class also only works if all computations have bounded length outside of the region. They fail to prove stability e.g. for the first example in Section 3 (a system with one location where the flow condition is given by $\dot{x} = -1$ and the region φ is given by $x \leq 0$).

The algorithm presented in [19] checks a condition related to region stability, namely *strong attraction*. Strong attraction implies but is not implied by region stability. The algorithm in [19] was not implemented. It would not make sense to apply the algorithm to the benchmarks in Section 3 since in all cases except four (examples 1, 2, 5, and 6) region stability but not strong attraction holds.

Our source-to-source transformation described in Section 5.1 is inspired by the by now classical technique of *seeding* for computing the binary reachability relation (originally for imperative programs), see e.g. [7,19,1,10]. The originality of the source-to-source transformation in Section 5.1 lies in the fact that the binary reachability relation generates the three specific kinds of snapshot sequences.

7 Conclusion

We have addressed a verification problem that has previously been out of reach for automatic methods. We have presented a method and tool (and implementation) for automatic proofs of region stability for hybrid systems. The formal basis of our approach is the new notion of snapshot sequences. This notion yields a characterization of region stability which can be tested effectively. We have demonstrated the practical potential of our approach by a number of experiments.

In the present version, the implementation of our tool applies a linear constraint solver [21,17] to check the well-foundedness of the binary relation between snapshots; this is appropriate since the PHAVer tool computes an overapproximation of the binary relation in the form of linear constraints. For future work, we plan to investigate the usefulness of well-foundedness checks by non-linear constraint solvers [9] in a different version of our tool.

Acknowledgements

We thank the anonymous referees for many helpful comments and suggestions.

References

1. Berdine, J., Chawdhary, A., Cook, B., Distefano, D., O'Hearn, P.: Variance Analysis from Invariance Analysis. In: POPL'07 (2007)
2. Branicky, M.S.: Stability of Hybrid Systems: State of the Art. In: CDC'97 (1997)
3. Branicky, M.S.: Multiple Lyapunov Functions and Other Analysis Tools for Switched and Hybrid Systems. In: Trans. on Automatic Control (1998)
4. Burchardt, H., Oehlerking, J., Theel, O.: The Role of State-space Partitioning in Automated Verification of Affine Hybrid System Stability. In: CCCT'05 (2005)
5. Burchardt, H., Oehlerking, J., Theel, O.: Towards Push-of-a-Button Stability Verification for Discrete-Time Hybrid Systems. In: PRDC'05 (2005)
6. Cousot, P., Cousot, R.: Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: POPL'77 (1977)
7. Cousot, P., Cousot, R.: "A la Floyd" Induction Principles for Proving Inevitability Properties of Programs. In: Algebraic methods in semantics, '86 (1986)
8. Cousot, P., Cousot, R.: Abstract Interpretation Frameworks. Journal of Logic and Computation (1992)
9. Cousot, P.: Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, Springer, Heidelberg (2005)
10. Cook, B., Podelski, A., Rybalchenko, A.: Termination proofs for systems code. In: PLDI'06 (2006)
11. Damm, W., Pinto, G., Ratschan, S.: Guaranteed Termination in the Verification of LTL Properties of Non-linear Robust Discrete Time Hybrid Systems. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, Springer, Heidelberg (2005)
12. Frehse, G.: PHAVer, http://www.cs.ru.nl/~goranf
13. Frehse, G., Krogh, B.H., Rutenbar, R.A.: Verifying Analog Oscillator Circuits Using Forward/Backward Abstraction Refinement. DATE'06 (2006)
14. Liberzon, D.: Switching in Systems and Control. Birkhäuser (2003)
15. Liberzon, D., Hespanha, J.P., Morse, A.S.: Stability of Switched Systems: a Lie-algebraic Condition. In: Systems and Control Letters (1999)
16. Lakshmikantham, V., Leela, S., Martynyuk, A.A.: Practical Stability of Nonliear Systems. World Scientific Pub. Co. Inc. (1990)
17. Podelski, A., Rybalchenko, A.: A Complete Method for the Synthesis of Linear Ranking Functions. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, Springer, Heidelberg (2004)
18. Podelski, A., Rybalchenko, A.: Transition invariants. In: LICS' 2004 (2004)
19. 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, Springer, Heidelberg (2006)
20. Podelski, A., Wagner, S., Method, A.: a Tool for Automatic Verification of Region Stability for Hybrid Systems. In: MPI-I-2007-2-001 technical report (2007) See, http://www.mpi-inf.mpg.de/~swagner/MPI-TechRep_07.pdf
21. Rybalchenko, A.: RankFinder, http://www.mpi-inf.mpg.de/~rybal/rankfinder

CSL Model Checking Algorithms for Infinite-State Structured Markov Chains*

Anne Remke and Boudewijn R. Haverkort

University of Twente
Design and Analysis of Communication Systems
Faculty for Electrical Engineering, Mathematics and Computer Science
{anne,brh}@cs.utwente.nl

Abstract. Jackson queueing networks (JQNs) are a very general class of queueing networks that find their application in a variety of settings. The state space of the continuous-time Markov chain (CTMC) that underlies such a JQN, is highly structured, however, of infinite size in as many dimensions as there are queues. We present CSL model checking algorithms for labeled JQNs. To do so, we rely on well-known product-form results for the steady-state probabilities in (stable) JQNs. The transient probabilities are computed using an uniformization-based approach. We develop a new notion of property independence that allows us to define model checking algorithms for labeled JQNs.

1 Introduction

Queueing networks have been used for about half a century now, for modeling and analyzing a wide variety of phenomena in computer, communication and logistics systems. Seminal work on queueing networks was done by Jackson in the 1950s [11,12]; in which he developed an important theorem that characterizes the steady-state (long-run) probabilities to reside in certain states in a restricted class of queueing networks (see the next section).

However, there are many phenomena in the above classes of systems that cannot be studied well using these long-run probabilities. In communication system models, the following situations reflect such cases: (i) what is the probability that starting from an initial empty system, within t time-units, at least k_i packets are buffered at queue i? (ii) what is the probability that starting from an overload situation, e.g., characterized by at least L packets at each queue, within t time-units, a low load situation, e.g., characterized by at most $l << L$ packets at each queue is reached again?

Clearly, the above sketched scenarios require more than just long-run probabilities. Since logics like CSL have been shown to be extremely helpful in specifying similar properties for finite CTMCs [2,3], we also pursue a CSL model checking procedure for the CTMCs that underlie so-called Jackson queueing networks

* The work presented in this paper has been performed in the context of the MC=MC project (612.000.311), financed by the Netherlands Organization for Scientific Research (NWO). The authors thank Lucia Cloth for fruitful discussions on the topic.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 336–351, 2007.

(JQNs). As we know from [3], for CSL model checking of CTMCs, we need to be able to compute both steady-state and transient state probabilities for all states, and for all possible starting states. The key issue lies in the fact that the CTMC underlying a JQN is infinite in as many dimensions as there are queues in the JQN. For the steady-state probabilities we can rely on the seminal work of Jackson, however, for the transient state probabilities, no results are readily available. Inspired by our recent work on CSL model checking for quasi-birth-death processes (QBDs), which form a different class of infinite-state CTMCs [15], we use an uniformization-based approach to compute just those transient state probabilities in JQNs that are needed to verify the validity of CSL properties. The highly structured state space allows us to conclude the validity of CSL properties for groups of states on the basis of the validity for a so-called representative state in such a group. This reduces the infinite number of state probabilities to be computed to a finite number.

We are not aware of any other work that addresses the model checking problem solved in this paper. However, there is *related work* on model checking infinite state systems, such as, e.g., Boucherie product processes [4], probabilistic lossy channel systems [18], regular model checking [1], recursive Markov chains [8] and probabilistic pushdown automata [5], as well as on transient analysis of queueing networks [14,10] and on transient analysis of infinite-state systems with uniformization [19,6]. None of this work, however, addresses the model checking questions that we address.

The rest of this paper is organized as follows. We introduce JQNs in Section 2 and discuss the form of the underlying state space and transition relation in detail in Section 3. We briefly rehearse the logic CSL in Section 4, before we discuss the model checking algorithms for all the CSL operators in Section 5. Note that we restrict ourselves to the time interval $[0, t]$ for the time-bounded until operator. Finally, Section 6 presents some conclusions. A running example to illustrate the key concepts and procedures is provided throughout the paper.

2 Jackson Queueing Networks

Jackson queueing networks (JQNs) consist of a number of interconnected queueing stations, numbered $1, \ldots, n$. At each individual queueing station i, jobs arrive with a negative exponential inter-arrival time distribution with rate λ_i, and the job service requirements are also negative exponentially distributed, however, with rate μ_i. There is a single server at each queueing station[1] Jobs arriving at a queue are served in *first come first served* (FCFS) order. Jobs arriving when the server is busy, are queued in an unbounded buffer. Each queue in a JQN behaves, in essence, as a simple so-called $M|M|1$ queue. We assume a never empty source from which customers originate and arrive at the JQN, and into which they disappear after having received their service. This environment is indexed 0 and the overall arrival process from the environment is a Poisson process with rate λ.

[1] This can be alleviated easily to m-servers.

A finite routing matrix $\mathbf{R} \in \mathbb{R}^{(n+1)\times(n+1)}$ contains the routing probabilities from queue i to queue j with $i, j \in \{0, 1, \cdots, n\}$: $r_{i,j} \in [0, 1]$. Note that $\sum_{j=0}^{n} r_{i,j} = 1$ for all i. In case $r_{i,j} = 0$ there is no routing from queue i to queue j and in case $r_{i,j} = 1$, j is the only output for queue i. The routing probability $r_{i,0}$ gives the probability that a job actually leaves the queueing network after completion at queue i and the routing probability $r_{0,j}$ gives the probability that an arriving customer is routed to queue j. Note that $r_{0,0} = 0$ by definition and that we do allow for self loops (e.g., $r_{i,i} > 0$ is allowed). A state in a JQN can be defined as $\overline{s} = (s_1, s_2, \cdots, s_n)$, where $s_i \geq 0$ represents the number of customers in queue i. A more precise discussion of the state space S is postponed to Section 3. For model checking purposes, we also need a state labeling. This leads us to the following definition.

Definition 1 (Jackson queueing networks)
A labeled Jackson queueing network **LJQN** $\mathcal{J}$ of order n (with $n \in \mathbb{N}^+$) is a tuple $(\lambda, \underline{\mu}, \mathbf{R}, L)$ with arrival rate λ, a vector of size n of service rates $\underline{\mu}$, a routing matrix $\mathbf{R} \in \mathbb{R}^{(n+1)\times(n+1)}$ and a labeling function L that assigns a set of valid atomic propositions from a fixed and finite set AP of atomic propositions to each state $\overline{s} = (s_1, s_2, \ldots, s_n)$. □

Restriction 1
In the following we will restrict ourselves to atomic propositions of the form $\bigwedge_{i=1}^{n}(s_i \lesseqgtr m_i)$ for $m_i \in \mathbb{N}$. This restricts the formulas we are able to check.

Example 1
In Figure 1(a) we present a LJQN with two queues that will serve as running example. The external arrival rate is λ, the vector of service rates is given as $\overline{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$ and the routing matrix is $\mathbf{R} = \begin{pmatrix} 0 & r_{0,1} & r_{0,2} \\ r_{1,0} & 0 & r_{1,2} \\ r_{2,0} & r_{2,1} & 0 \end{pmatrix}$. The labeling L will be introduced later.

Definition 2 (Traffic equations)
The overall flow of jobs through each queue j is given as: $\lambda_j = \lambda r_{0,j} + \sum_{i=1}^{n} \lambda_i r_{i,j}$, for $j = 1, \ldots, n$. These equations are called *(first-order) traffic equations.* □

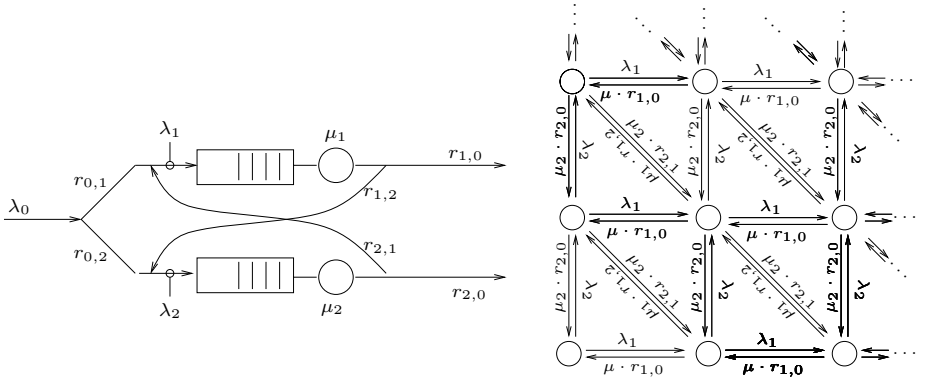
Definition 3 (Utilization)
The utilization per queue i is defined as $\rho_i = \frac{\lambda_i}{\mu_i}$. □

Restriction 2
In case all $\rho_i = \frac{\lambda_i}{\mu_i} < 1$, the QN is said to be stable. That is the number of arriving jobs per unit time is smaller than the amount of jobs that each queue can handle per unit time. This guarantees that the queue will not build up infinitely. In the following we restrict ourselves to stable LJQNs, to be able to compute steady-state probabilities.

The long-run probability that s customers are presently in a single $M|M|1$ queue, that is, the so-called *steady-state probability*, for a single stable $M|M|1$ queue (with $\rho = \frac{\lambda}{\mu} < 1$) is: $\Pr\{S = s\} = (1-\rho)\rho^s$, where S is the random variable indicating the number of customers in the queue [13]. In [11,12], Jackson proved the following theorem:

Theorem 1 (Jackson)
The overall steady-state probability distribution under restriction 2 (stability) is the product of the per-queue steady-state probability distributions, where the queues can be regarded as if operating independently from each other:

$$\Pr\{\overline{S} = \overline{s}\} = \prod_{i=1}^{n}(1-\rho_i)\rho_i^{s_i}, \qquad \overline{s} = (s_1, \ldots, s_n) \in \mathsf{S}. \tag{1}$$



(a) two queues with feedback (b) underlying state-space

Fig. 1. JQN with two queues

3 State Space, Transitions, Independence and Paths

This section addresses in detail the underlying state space of a LJQN. In Section 3.1 the underlying infinite state structured Markov chain is described. Section 3.2 explains how the infinite state space can be structured using atomic propositions and a new notion of independence is introduced. In Section 3.3 we define paths and transient as well as steady-state probabilities on LJQNs.

3.1 Infinite State Structured Markov Chain

The underlying state space of a LJQN $\mathcal{J}$ of order n is a highly structured labeled infinite-state continuous-time Markov chain with state space $\mathsf{S} = \mathbb{N}^n$ that is infinite in n dimensions. Every state $\overline{s} \in \mathsf{S}$ can be represented as an n-tuple $\overline{s} = (s_1, s_2, \cdots, s_n)$, with $s_i \geq 0$. The labeling function $L : \mathsf{S} \to 2^{AP}$ on the

state space then assigns from the set AP of atomic propositions the set of valid atomic propositions to each state. The state $\hat{s} = (0, \ldots, 0)$ is called origin. As the number of customers in queue i is restricted to $s_i \geq 0$, for all $i \in \{1, \ldots, n\}$, the underlying state-space is limited towards the origin in every dimension. An n dimensional state space S is bounded by n so-called *boundary hyperplanes* of dimension $n-1$. Note that these boundary hyperplanes consist of an infinite number of states for $n \geq 2$. State changes may occur due to an arrival at queue i from the environment or a departure to the environment from queue i, or by routing of a customer from queue i to queue j for $1 \leq i, j \leq n$. By adding a state change vector to the source state, the destination state is defined: an arrival at queue i is denoted by the vector $\overline{a}_i$, a departure from queue i is denoted by the vector $\overline{d}_i$, and a job routing from queue i to queue j is denoted as vector $\overline{f}_{i,j}$, with:

$$\overline{a}_i = \begin{cases} a_j = 1 & i = j, \\ a_j = 0 & j \neq i. \end{cases} \quad \overline{d}_i = \begin{cases} d_j = -1 & j = i, \\ d_j = 0 & j \neq i. \end{cases} \quad \overline{f}_{i,j} = \begin{cases} f_k = -1 & k = i, \\ f_k = 1 & k = j, \\ f_k = 0 & \text{otherwise.} \end{cases}$$

Note that an arrival is always possible, the new state is then defined as $\overline{s}' = \overline{s} + \overline{a}_i$. A departure or a job routing from queue i is only possible, when there is at least one customer in queue i; in this case the new state $\overline{s}'$ is computed as: $\overline{s}' = \overline{s} + \overline{d}_i$ or $\overline{s}' = \overline{s} + \overline{f}_{i,j}$, respectively.

Definition 4 (Generator function)
The rate for a state change from a state $\overline{s}$ to another state $\overline{s}'$ within the infinite state space S is given by the generator function $\mathbf{G}(\overline{s}, \overline{s}') : \mathsf{S} \times \mathsf{S} \to \mathbb{R}^+$, for $\overline{s} \neq \overline{s}'$, as follows:

cause	restriction	state change	$\mathbf{G}(\overline{s}, \overline{s}')$
arrival	none	$\overline{s}' = \overline{s} + \overline{a}_i$	$\lambda \cdot r_{0,i}$
departure	$s_i > 0$	$\overline{s}' = \overline{s} + \overline{d}_i$	$\mu_i \cdot r_{i,0}$
routing	$s_i > 0$	$\overline{s}' = \overline{s} + \overline{f}_{i,j}$	$\mu_i \cdot r_{i,j}$

and $G(\overline{s}, \overline{s}') = 0$ for $\overline{s} \neq \overline{s}'$ in all other cases. $\mathbf{G}(\overline{s}, \overline{s})$ is defined as the negative sum over all possible outgoing rates from $\overline{s}$, that is

$$\mathbf{G}(\overline{s}, \overline{s}) = -\left(\lambda + \sum_{i=1}^{n} \mu_i \cdot \mathbf{1}_{s_i > 0}\right),$$

where the indicator function $\mathbf{1}_{s_i>0}$ returns 1 if $s_i > 0$, and 0 otherwise. □

Example 2
Figure 1(b) shows the underlying state space of the LJQN from Example 1 that is infinite in two dimensions. Arrivals occur in both dimensions with rate $\lambda \cdot r_{0,1}$ and $\lambda \cdot r_{0,2}$, departures happen from both dimensions with rates $\mu_1 \cdot r_{1,0}$ and $\mu_2 \cdot r_{2,0}$, respectively, and jobs are routed from queue 1 to queue 2 with rate $\mu_1 \cdot r_{1,2}$ and from queue 2 to queue 1 with rate $\mu_2 \cdot r_{2,1}$.

3.2 Independence of Atomic Propositions

Recall that we restrict ourselves to atomic propositions of the form $\bigwedge_{i=1}^{n}(s_i \lesseqgtr m_i)$ for $m_i \in \mathbb{N}$. Due to this restriction, the validity of an atomic proposition does not change anymore for $s_i \geq m_i$ onwards for dimension i. Hence, we can define the so-called *independence vector* $\overline{m} = (m_1, \ldots, m_n)$ and call the atomic proposition *independent as of* $\overline{m}$. For the set of states $\{\overline{s} \in \mathsf{S} \mid \forall i(s_i \geq m_i)\}$ the validity of the atomic proposition remains the same.

The state space can be partitioned into a finite set of boundary states S_b and a finite number of infinite representative sets of states (denoted $S_{\overline{r}}$) such that the validity of an atomic proposition $ap \in AP$ does not change any more in this set. We choose a *representative state* $\overline{r}$ for each of these infinite sets $S_{\overline{r}}$ such that for all $\overline{s} \in S_{\overline{r}}$ the labeling does not change: $L(\overline{r}) = L(\overline{s})$, for all $\overline{s} \in S_{\overline{r}}$. In general, in an n-dimensional LJQN, there are n types of representative sets that account for 1 up to n infinite dimensions. A representative set is called infinite in dimension i if and only if $r_i = m_i$, and restricted in dimension i otherwise. In case a representative state $\overline{r}$ is infinite in k dimensions it represents a k dimensional set $S_{\overline{r}}$, such that

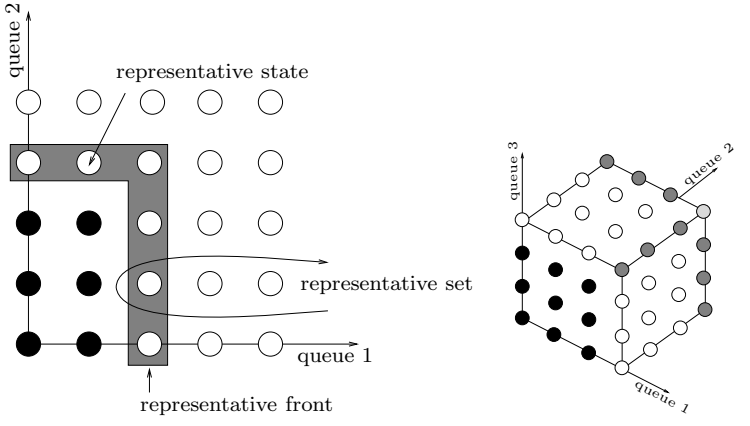
$$\overline{s} \in S_{\overline{r}} \Leftrightarrow \begin{cases} s_i \geq r_i & \text{iff } r_i = m_i, \\ s_i = r_i & \text{otherwise.} \end{cases}$$

Hence, a state $\overline{s}$ belongs to $S_{\overline{r}}$ when it takes the same value as $\overline{r}$ in the restricted dimensions and any value $\geq r_i$ in the infinite dimensions. For atomic propositions, the origin $\hat{s}$ represents the finite set of boundary states that is defined as $S_b = \{\overline{s} \in \mathsf{S} \mid s_i < m_i, \forall i \in \{1, \ldots, n\}\}$. The finite union of all representative states is called *representative front* and defined as $R(\overline{m}) = \{\overline{r} \in \mathsf{S} \mid \exists i : (r_i = m_i) \wedge (\forall j \neq i : (r_j \leq m_j))\}$. Note that $\mathsf{S} = S_b \cup \bigcup_{\overline{r} \in R(\overline{m})} S_{\overline{r}}$. For atomic propositions the representative front can be made smaller, however, for model checking CSL properties in general we need the full representative front as defined above.

Example 3
Suppose we define the atomic proposition $\text{ap}_1 = (s_1 \geq 2) \wedge (s_2 \geq 3)$ for the LJQN from Example 1; the white states in Figure 2(a) then depict those states where ap_1 is valid. The representative front for ap_1 is formed by the states in the grey polygon: $(0, 3)$ accounts for the states $(0, j)$, with $j \geq 3$, and $(1, 3)$ represents the states $(1, j)$ with $j \geq 3$. For $i \geq 2$, $(2, 0)$ represents $(i, 0)$, $(2, 1)$ represents $(i, 1)$ and $(2, 2)$ represents $(i, 2)$, respectively. These five representative states all account for a one dimensional set of states. The representative state $(2, 3)$ accounts for the two-dimensional set of states $S_{(2,3)} = \{\overline{s} \in \mathsf{S} \mid s_1 > 2 \wedge s_2 > 3\}$. The black states belong the the boundary set S_b with representative $(0, 0)$.

Example 4
Figure 2(b) shows the representative front for the atomic proposition $\text{ap}_2 = (s_1 < 3) \wedge (s_2 < 3) \wedge (s_3 < 3)$ in a LJQN of dimension three. The atomic proposition is valid in the black states only (of the 27 only 9 are visible). All the remaining depicted states are representative states. We have different types of

representative states: the white ones represent a set of states that is infinite in one dimension, the grey ones represent a set that is infinite in two dimensions and the light grey one $(3, 3, 3)$ represents a set that is infinite in three dimensions.



(a) 2-dim. state space **(b)** 3-dim state space

Fig. 2. Representative front of states for independent atomic propositions

3.3 Paths and Probabilities

In what follows, we first present the notion of path, before we define steady-state and transient state probabilities.

Definition 5 (Paths)
An *infinite path* σ is a sequence $\overline{s}_0 \xrightarrow{t_0} \overline{s}_1 \xrightarrow{t_1} \overline{s}_2 \xrightarrow{t_2} \ldots$ with, for $i \in \mathbb{N}$, $\overline{s}_i \in \mathsf{S}$ and $t_i \in \mathbb{R}_{>0}$ such that $\mathbf{G}(\overline{s}_i, \overline{s}_{i+1}) > 0$ for all i. A *finite path* σ of length $l+1$ is a sequence $\overline{s}_0 \xrightarrow{t_0} \overline{s}_1 \xrightarrow{t_1} \ldots \overline{s}_{l-1} \xrightarrow{t_{l-1}} \overline{s}_l$ such that $\overline{s}_l$ is absorbing[2], and $\mathbf{G}(\overline{s}_i, \overline{s}_{i+1}) > 0$ for all $i < l$. For an infinite path σ, $\sigma[i] = \overline{s}_i$ denotes for $i \in \mathbb{N}$ the $(i+1)$st state of path σ. The time spent in state $\overline{s}_i$ is denoted by $\delta(\sigma, i) = t_i$. Moreover, with i the smallest index with $t \leq \sum_{j=0}^{i} t_j$, let $\sigma@t = \sigma[i]$ be the state occupied at time t. For finite paths σ with length $l+1$, $\sigma[i]$ and $\delta(\sigma, i)$ are defined in the way described above for $i < l$ only and $\delta(\sigma, l) = \sigma[l] = \infty$ and $\delta@t = \overline{s}_l$ for $t > \sum_{j=0}^{l-1} t_j$. $Path^{\mathcal{J}}(\overline{s})$ is the set of all finite and infinite paths in the LJQN $\mathcal{J}$ that start in state $\overline{s}$ and $Path^{\mathcal{J}}$ includes all (finite and infinite) paths of the LJQN $\mathcal{J}$. □

As for finite CTMCs, a probability measure on paths can now be defined depending on the starting state [3]. Starting from there, two different types of state probabilities can be distinguished.

[2] A state $\overline{s}$ is called absorbing if for all $\overline{s}'$ the rate $\mathbf{G}(\overline{s}, \overline{s}') = 0$.

The transient state probability is a time-dependent measure that considers the LJQN at a given time instant t. The probability to be in state $\overline{s}'$ at time instant t, given initial state $\overline{s}$, is denoted as $\mathbf{V}^{\mathcal{J}}(\overline{s},\overline{s}',t) = \Pr\{\sigma \in Path^{\mathcal{J}}(\overline{s}) \mid \sigma@0 = \overline{s} \wedge \sigma@t = \overline{s}'\}$. The transient probabilities are characterized by a linear system of differential equations of infinite size. Let $\mathbf{V}(t)$ be the matrix of transient state probabilities at time t for all possible starting states $\overline{s}$ and for all possible goal states $\overline{s}'$ (we omit the superscript $\mathcal{J}$ for brevity here), then we have $\mathbf{V}'(t) = \mathbf{V}(t) \cdot \mathbf{G}$, given $\mathbf{V}(0) = \mathbf{I}$. An efficient method to compute the transient probabilities will be discussed in Section 5.4.

The steady-state probabilities to be in state $\overline{s}'$, given initial state $\overline{s}$, are defined as $\pi^{\mathcal{J}}(\overline{s},\overline{s}') = \lim_{t\to\infty} \mathbf{V}^{\mathcal{J}}(\overline{s},\overline{s}',t)$, and indicate the probabilities to be in some state $\overline{s}'$ "in the long run". If steady-state is reached, the above mentioned derivatives $\mathbf{V}'(t)$ will approach zero. As we require stable queues the underlying state space of the LJQN is ergodic and, the initial state does not influence the steady-state probabilities (we therefore write $\pi(\overline{s}')$ instead of $\pi(\overline{s},\overline{s}')$ for brevity). In the context of LJQNs the steady-state probabilities per state can be computed using the product-form of Theorem 1.

4 The logic CSL

We use the logic CSL [3] to express properties for LJQNs. The syntax and semantics are the same as for finite CTMCs, with the only difference that we now interpret the formulas over states and paths in LJQNs. Let $p \in [0,1]$ be a real number, $\bowtie \in \{\leq, <, >, \geq\}$ a comparison operator, $t_1, t_2 \in \mathbb{R}^+$ real numbers and AP a set of atomic propositions with $ap \in AP$. CSL state formulas Φ are defined by

$$\Phi ::= \mathtt{tt} \mid ap \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathcal{S}_{\bowtie p}(\Phi) \mid \mathcal{P}_{\bowtie p}(\phi),$$

where ϕ is a path formula constructed by

$$\phi ::= \mathcal{X}^{[t_1,t_2]}\Phi \mid \Phi\, \mathcal{U}^{[t_1,t_2]}\Psi.$$

For a CSL state formula Φ and a LJQN $\mathcal{J}$, the satisfaction set $Sat(\Phi)$ contains all states of $\mathcal{J}$ that fulfill Φ and is computed with a recursive descent procedure over the parse tree of Φ, as for CTL [7]. Satisfaction is stated in terms of a satisfaction relation $\models$. The relation $\models$ for states and CSL state formulas is defined as:

$$\begin{array}{ll}
\overline{s} \models \mathtt{tt} \text{ for all } \overline{s} \in \mathsf{S}, & \overline{s} \models \Phi \wedge \Psi \quad \text{iff } \overline{s} \models \Phi \text{ and } \overline{s} \models \Psi, \\
\overline{s} \models ap \text{ iff } ap \in L(\overline{s}), & \overline{s} \models \mathcal{S}_{\bowtie p}(\Phi) \text{ iff } \pi^{\mathcal{J}}(\overline{s}, Sat(\Phi)) \bowtie p, \\
\overline{s} \models \neg\Phi \text{ iff } \overline{s} \not\models \Phi, & \overline{s} \models \mathcal{P}_{\bowtie p}(\phi) \text{ iff } Prob^{\mathcal{J}}(\overline{s}, \phi) \bowtie p,
\end{array}$$

where $\pi^{\mathcal{J}}(\overline{s}, Sat(\Phi)) = \sum_{\overline{s}' \in Sat(\Phi)} \pi^{\mathcal{J}}(\overline{s},\overline{s}')$, and $Prob^{\mathcal{J}}(\overline{s},\phi)$ describes the probability measure of all paths $\sigma \in Path(\overline{s})$ that satisfy ϕ when starting in state $\overline{s}$, that is, $Prob^{\mathcal{J}}(\overline{s},\phi) = \Pr\{\sigma \in Path^{\mathcal{J}}(\overline{s}) \mid \sigma \models \phi\}$. The steady-state operator $\mathcal{S}_{\bowtie p}(\Phi)$ denotes that the steady-state probability for Φ-states meets the bound p.

$\mathcal{P}_{\bowtie p}(\phi)$ asserts that the probability measure of the paths satisfying ϕ meets the bound p. The next operator $\mathcal{X}^{[t_1,t_2]}\Phi$ states that a transition to a Φ-state is made between t_1 and t_2. The until operator $\Phi\ \mathcal{U}^{[t_1,t_2]}\Psi$ asserts that Ψ is satisfied at some time instant in between $[t_1, t_2]$ and that at all preceding time instants Φ holds. The relation $\models$ for paths and CSL path formulas is defined as:

$$\sigma \models \mathcal{X}^{[t_1,t_2]}\Phi \qquad \text{iff } \sigma[1] \text{ is defined and } \sigma[1] \models \Phi \text{ and } t_1 \leq \delta(\sigma, 0) \leq t_2,$$

$$\sigma \models \Phi\ \mathcal{U}^{[t_1,t_2]}\Psi \qquad \text{iff } \exists t(t_1 \leq t \leq t_2)\ (\sigma@t \models \Psi \wedge (\forall t' \in [0, t)(\sigma@t' \models \Phi))).$$

From [15] we know that CSL formulas are not level independent on QBDs in general, even if the atomic propositions are level independent. However, their validity does not change arbitrarily between levels. In the following we will show that for CSL formulas on LJQNs we can also find a state from which the validity of the CSL formula does not change anymore.

Definition 6 (Independence of CSL formulas)
Let $\mathcal{J}$ be a LJQN of order n. A CSL state formula Φ is *independent* as of $\overline{m}$ if and only if there exists a finite *representative front* $R(\overline{m}) = \{\overline{r} \in \mathsf{S} \mid \exists i(r_i = m_i) \wedge (\forall j \neq i(r_j \leq m_j))\}$ such that for all $\overline{r} \in R(\overline{m})$ and for all $\overline{s} \in S_{\overline{r}}$ it holds that $\overline{r} \models \Phi \Leftrightarrow \overline{s} \models \Phi$. □

The following propositions states, under the assumption of independent atomic propositions as of $\overline{m}$, that such a finite representative front $R(\overline{m}')$ exists for any CSL state formula. We will justify this proposition inductively over the structure of the logic in Section 5.

Proposition 1
Let $\mathcal{J}$ be a LJQN of order n with independent atomic propositions as of $\overline{m}$ and let Φ be a CSL state formula other than $\mathcal{P}_{\bowtie p}(\Phi\ \mathcal{U}^{[t_1,t_2]}\Psi)$. Then there exists a finite *representative front* $R(\overline{m}')$ such that the validity of Φ does not change within each subset $S_{\overline{r}}$ for $\overline{r} \in R(\overline{m}')$ in $\mathcal{J}$. For the until operator $\mathcal{P}_{\bowtie p}(\Phi\ \mathcal{U}^{I}\Psi)$ we assume that for no state $\overline{s}$ the probability measure is exactly equal to p. Under this assumption, there exists a vector $\overline{m}'$, such that $\mathcal{P}_{\bowtie p}(\Phi\ \mathcal{U}^{I}\Psi)$ is independent as of $\overline{m}'$ in $\mathcal{J}$. □

For a CSL formula that is independent as of $\overline{m}$, the satisfaction set can be considered as the union of the *boundary satisfaction set* $Sat^{S_b}(\Phi) = S_b \cap Sat(\Phi)$ and the *representative satisfaction front* $Sat^{R(\overline{m})}(\Phi) = R(\overline{m}) \cap Sat(\Phi)$. The representative state $\overline{r} \in R(\overline{m})$ then represent the remaining infinite state space.

5 Model Checking Algorithms

In this section we present model checking algorithms for CSL. In Section 5.1 we explain how independence changes when applying logical operators. We explain how to model check the steady-state operator in Section 5.2, the next operator in 5.3 and the until operator in Section 5.4, including a discussion on how to compute transient probabilities in LJQNs.

5.1 Logical Operators

The model checking procedure for logical operators is the same as for finite CTMCs. The only thing we need to take care of is how independence changes. Negating a CSL formula does not change its independence. For a CSL formula Φ that is independent as of $\overline{m}$ the negation $\neg\Phi$ is also independent as of $\overline{m}$. However, combining a CSL formula Φ that is independent as of $\overline{m}^{\Phi}$ with a CSL formula Ψ that is independent as of $\overline{m}^{\Psi}$ with conjunction, changes independence depending on the structure of Φ and Ψ. In any case, we can state that $\Phi \wedge \Psi$ is independent as of $\overline{m} = \max\{\overline{m}^{\Phi}, \overline{m}^{\Psi}\}$, where we choose the maximum of m_i^{Φ} and m_i^{Ψ} in every dimension i. Note that this new independence vector $\overline{m}$ might be too pessimistic, depending on the structure of Φ and Ψ.

5.2 Steady-State Operator

Theorem 1 states the steady-state probabilities in a JQN. A state s satisfies $\mathcal{S}_{\bowtie p}(\Phi)$ if the sum of the steady-state probabilities of all Φ-states reachable from s meets the bound p. Since a JQN is by definition ergodic, the steady-state probabilities are independent of the starting state. It follows that either all states satisfy a steady-state formula or none of the states does, which implies that a steady-state formula is always independent as of $\overline{m} = (0, \ldots, 0)$. We sum the steady state probabilities of all states that satisfy Φ by summing over all states $\overline{s} \in S_{\overline{r}}$ for $\overline{r} \models \Phi$ and over all states from S_b that satisfy Φ:

$$\overline{s} \models \mathcal{S}_{\bowtie p}(\Phi) \Leftrightarrow \sum_{\substack{\overline{s} \in S_b, \\ \overline{s} \in Sat(\Phi)}} p(\overline{s}) + \sum_{\substack{\overline{r} \in R(\overline{m}), \\ \overline{r} \in Sat(\Phi)}} \sum_{\overline{s} \in S_{\overline{r}}} p(\overline{s}) \bowtie p \tag{2}$$

We obtain $Sat(\mathcal{S}_{\bowtie p}(\Phi)) = \mathsf{S}$, if the accumulated steady-state probability meets the bound p otherwise $Sat(\mathcal{S}_{\bowtie p}(\Phi)) = \varnothing$. In case the representative state $\overline{r} \in Sat(\Phi)$, all states $\overline{s} \in S_{\overline{r}}$ are in $Sat(\Phi)$. The accumulated steady-state probability for all states $\overline{s} \in S_{\overline{r}}$ is given by the following expression:

$$\sum_{\overline{s} \in S_{\overline{r}}} p(\overline{s}) = \prod_{i=1}^{n} \Omega(i), \text{ with } \Omega(i) = \begin{cases} (1-\rho_i)\rho_i^{r_i} & \text{for } r_i \neq m_i, \\ \rho_i^{m_i} & \text{for } r_i = m_i. \end{cases} \tag{3}$$

In this expression we distinguish between the finite and the infinite dimensions of a representative state $\overline{r}$. In a finite dimension i we multiply with $(1-\rho_i)\rho_i^{r_i}$ and in an infinite dimension i we multiply with $\rho_i^{m_i}$.

Proof 1 (Accumulated steady-state probability)
Applying Jackson's Theorem, the accumulated steady-state probability is given by

$$\sum_{\overline{s} \in S_{\overline{r}}} p(\overline{s}) = \sum_{\overline{s} \in S_{\overline{r}}} \left(\prod_{i=1}^{n} (1-\rho_i)\rho_i^{s_i} \right). \tag{4}$$

Recall that

$$\overline{s} \in S_{\overline{r}} \Leftrightarrow \begin{cases} s_i \geq r_i & \text{iff } r_i = m_i, \\ s_i = r_i & \text{otherwise.} \end{cases}$$

According to Jackson's Theorem we can consider the dimensions independently from each other. Hence, for every dimension $i = 1, \ldots, n$, a state $\overline{s} \in S_{\overline{r}}$ may take all values $j \geq m_i$ in case $r_i = m_i$ and the value r_i in case $r_i \neq m_i$. Thus,

$$\sum_{\overline{s} \in S_{\overline{r}}} p(\overline{s}) = \prod_{i=1}^{n} \Omega(i), \text{ with } \Omega(i) = \begin{cases} (1-\rho_i)\rho_i^{r_i} & \text{for } r_i \neq m_i, \\ \sum_{j=m_i}^{\infty} (1-\rho_i)\rho_i^j & \text{for } r_i = m_i. \end{cases} \tag{5}$$

The infinite sum can be rewritten

$$\sum_{j=m_i}^{\infty} (1-\rho_i)\rho_i^j = (1-\rho_i) \sum_{j=m_i}^{\infty} \rho_i^j = (1-\rho_i)\frac{\rho_i^{m_i}}{1-\rho_i} = \rho_i^{m_i}$$

and replaced to match (3):

$$\Omega(i) = \begin{cases} (1-\rho_i)\rho_i^{r_i} & \text{for } r_i \neq m_i, \\ \rho_i^{m_i} & \text{for } r_i = m_i. \end{cases}$$

□

Example 5
We want to check the CSL formula $\mathcal{S}_{\bowtie p}((s_1 \geq 2) \wedge (s_2 \geq 3))$. Recall from Example 3 that all states $\overline{r} \in R((2,3)) = \{(0,3),(1,3),(2,0),(2,1),(2,3)\}$ satisfy $ap_1 = (s_1 \geq 2) \wedge (s_2 \geq 3)$ and the boundary states do not satisfy ap_1. Using (2) and (3) and accumulating the probabilities for all representative states $\overline{r} \in R((2,3))$ we obtain

$$\begin{aligned} &\overline{s} \models \mathcal{S}_{\bowtie p}((s_1 \geq 2) \wedge (s_2 \geq 3)) \Leftrightarrow \\ &\left((1-\rho_1)\rho_2^3 + (1-\rho_1)\rho_1\rho_2^3 + \rho_1^2(1-\rho_2) + \rho_1^2(1-\rho_2)\rho_2 + \rho_1^2\rho_2^3\right) \bowtie p. \end{aligned} \tag{6}$$

5.3 Time-Bounded Next Operator

The time-bounded next operator for LJQNs is computed just as for QBDs [15]. Recall that a state $\overline{s}$ satisfies $\mathcal{P}_{\bowtie p}(\mathcal{X}^{[t_1,t_2]}\Phi)$ if the one-step probability to reach a state that fulfills Φ within a time $t \in [t_1, t_2]$, outgoing from $\overline{s}$ meets the bound p. The possibly infinite summation over all Φ-states can be truncated by only considering those Φ-states that can be reached in one step from $\overline{s}$. We define the set of states that is reachable in one step from state $\overline{s}$ as $B_{\overline{s}} = \{\overline{s}' \in S | G(\overline{s}, \overline{s}') > 0\}$. Note that $B_{\overline{s}}$ is always finite. We then have:

$$\begin{aligned} \overline{s} \models \mathcal{P}_{\bowtie p}(\mathcal{X}^{[t_1,t_2]}\Phi) \;&\Leftrightarrow\; \Pr\{\sigma \in Path(\overline{s}) \mid \sigma \models \mathcal{X}^{[t_1,t_2]}\Phi\} \;\bowtie p \\ \Leftrightarrow \left(\left(e^{\mathbf{G}(\overline{s},\overline{s})\cdot t_1} - e^{\mathbf{G}(\overline{s},\overline{s})\cdot t_2}\right) \cdot \sum_{\substack{\overline{s}' \in Sat(\Phi) \cap B_{\overline{s}} \\ \overline{s} \neq \overline{s}'}} \frac{\mathbf{G}(\overline{s},\overline{s}')}{-\mathbf{G}(\overline{s},\overline{s})} \right) &\bowtie p, \end{aligned} \tag{7}$$

where $e^{G(\overline{s},\overline{s})\cdot t_1} - e^{G(\overline{s},\overline{s})\cdot t_2}$ is the probability of residing in $\overline{s}$ for a time $t \in [t_1, t_2]$, and $\frac{G(\overline{s},\overline{s}')}{-G(\overline{s},\overline{s})}$ specifies the probability to step from state $\overline{s}$ to state $\overline{s}'$, provided a step takes place.

Now, let the inner formula Φ be independent as of $\overline{m}$. Hence, the validity of Φ might be different for all states $\overline{s} \in S_b$. Therefore, the representative states $\overline{r} \in R(\overline{m})$ may satisfy $\mathcal{P}_{\bowtie p}(\mathcal{X}^{[t_1,t_2]}\Phi)$, whereas the remaining states $\overline{s} \in S_{\overline{r}}$ do not necessarily satisfy $\mathcal{P}_{\bowtie p}(\mathcal{X}^{[t_1,t_2]}\Phi)$, since S_b is reachable in one step. However, from $\overline{m} + \overline{1}$ (with $\overline{1} = (1, 1, \ldots, 1, 1)$) onwards, with one step only states with equivalent Φ validity can be reached. Thus, in case the inner formula Φ is independent as of $\overline{m}$, $\mathcal{P}_{\bowtie p}(\mathcal{X}^{[t_1,t_2]}\Phi)$ is independent as of $\overline{m} + \overline{1}$. For the construction of the satisfaction set of such a formula we have to compute explicitly the satisfying states in S_b. $Sat^{R(\overline{m}+\overline{1})}(\mathcal{P}_{\bowtie p}(\mathcal{X}^{[t_1,t_2]}\Phi))$ then provides the validity of $\mathcal{P}_{\bowtie p}(\mathcal{X}^{[t_1,t_2]}\Phi)$ for the remaining infinite state space $\mathsf{S} \setminus S_b$.

5.4 Until Operator for $I = [0, t]$

For model checking $\mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^I \Psi)$ we adopt the general approach for finite CTMCs [3] and QBDs [15]. Recall that the CSL path formula $\varphi = \Phi\, \mathcal{U}^I \Psi$ is valid if a Ψ-state is reached on a path during the time interval $[t_1, t_2]$ via only Φ-states. We restrict ourselves to intervals of the form $I = [0, t]$. The future behavior of the LJQN is then irrelevant for the validity of φ, as soon as a Ψ-state is reached. Thus all Ψ-states can be made absorbing without affecting the satisfaction set of formula φ. On the other hand, as soon as a $(\neg\Phi \wedge \neg\Psi)$-state is reached, φ will be invalid, regardless of the future evolution.

As a result of the above consideration, we may switch from checking the LJQN $\mathcal{J}$ to checking a new, derived, LJQN, denoted as $\mathcal{J}[\Psi][\neg\Phi \wedge \neg\Psi] = \mathcal{J}[\neg\Phi \vee \Psi]$, where all states in the underlying Markov chain that satisfy the formula in square brackets are made absorbing. The generator function $\tilde{\mathbf{G}}(\overline{s}, \overline{s}')$ for $\mathcal{J}[\neg\Phi \vee \Psi]$ is then defined as

$$\tilde{\mathbf{G}}(\overline{s}, \overline{s}') = \begin{cases} \mathbf{G}(\overline{s}, \overline{s}'), & \overline{s} \not\models \neg\Phi \vee \Psi, \\ 0, & \text{otherwise}, \end{cases} \tag{8}$$

for $\overline{s} \neq \overline{s}'$. $\tilde{\mathbf{G}}(\overline{s}, \overline{s})$ is adapted accordingly. Model checking a formula involving the until operator then reduces to calculating the transient probabilities $\pi^{\mathcal{J}[\neg\Phi\vee\Psi]}(\overline{s}, \overline{s}', t)$ for all Ψ-states $\overline{s}'$. Exploiting the structure of LJQNs yields

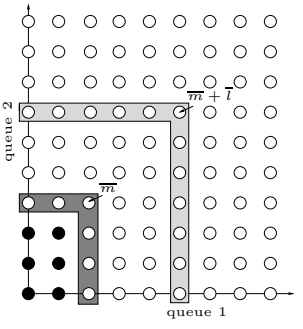
$$\begin{aligned} & s \models \mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^{[0,t]}\Psi) \Leftrightarrow Prob^{\mathcal{J}}(s, \Phi\, \mathcal{U}^{[0,t]}\Psi) \bowtie p \\ \Leftrightarrow & \left(\sum_{\overline{s}' \in Sat^{S_b}(\Psi)} \pi^{\mathcal{J}[\neg\Phi\vee\Psi]}(\overline{s}, \overline{s}', t) + \sum_{i=0}^{\infty} \sum_{s' \in Sat^{R(\overline{m}+\overline{i})}(\Psi)} \pi^{\mathcal{J}[\neg\Phi\vee\Psi]}(\overline{s}, \overline{s}', t) \right) \bowtie p. \end{aligned} \tag{9}$$

The transient probabilities are accumulated for the Ψ states in S_b and for the Ψ states in the representative front $R(\overline{m} + \overline{i})$ for $i \in \mathbf{N}$. Note that these representative fronts are situated in layers around S_b and cover the infinite state space $\mathsf{S} \setminus S_b$ for $i \in \mathbf{N}$.

In [15] we have shown how transient probabilities can be computed for QBDs using uniformization [9] by considering only a finite fraction of the infinite state

space. As standard property of uniformization, the finite time bound t is transformed to a finite number of steps l [9]. To do the same for LJQNs, first the probability function $\mathbf{P}(\overline{s},\overline{s}')$ (or for $\tilde{\mathbf{P}}$) for the embedded DTMC is defined as $\mathbf{P}(\overline{s},\overline{s}') = \frac{\mathbf{G}(\overline{s},\overline{s}')}{\nu}$ for $\overline{s} \neq \overline{s}'$ and $\mathbf{P}(\overline{s},\overline{s}) = \frac{\mathbf{G}(\overline{s},\overline{s})}{\nu} + 1$. The uniformization constant ν must be at least equal to the maximum of the negative rates $\mathbf{G}(\overline{s},\overline{s})$; for LJQNs, the value $\nu = \lambda + \sum_{i=1}^{n} \mu_i$ suffices. For an allowed maximal numerical error ε, uniformization requires a finite number l of steps (state changes) to be taken into account in order to compute the transient probabilities; l can be computed *a priori*, given ε, ν and t. Summarizing, we have obtained the following result:

If $\neg\Phi \vee \Psi$ is independent as of $\overline{m}$ then using uniformization with l steps, we obtain the same transient probabilities for states $\overline{s} \in S_{\overline{r}}$, with $\overline{r} \in R(\overline{m}+\overline{l})$, since from all states $\overline{s} \in S_{\overline{r}}$, only corresponding states can be reached when taking l steps in the LJQN.



Example 6
The figure to the left shows the finite fraction of the infinite state space that is needed to compute the validity of $\mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^I \Psi)$. Starting from every representative state $\overline{r} \in R(\overline{m}+\overline{l})$, still l steps can be undertaken in every direction without reaching the boundary set S_b. The total amount of states we have to consider equals $(m_1+l)\cdot(m_2+l)$, from which m_1+m_2+2l-1 are representative states.

For all states $s \in \mathsf{S}$, we add the computed transient probabilities to reach any Ψ-state and check whether the accumulated probability meets the bound p. We define the accumulated probability for up to l steps in the uniformized Markov chain as:

$$\tilde{\pi}^{(l)} = \sum_{\overline{s}' \in Sat^{S_b}(\Psi)} \pi^{\mathcal{J}[\neg\Phi\vee\Psi]}(\overline{s},\overline{s}',t) + \sum_{i=0}^{l-1} \sum_{s' \in Sat^{R(\overline{m}+\overline{i})}(\Psi)} \pi^{\mathcal{J}[\neg\Phi\vee\Psi]}(\overline{s},\overline{s}',t) \quad (10)$$

Note that the above expression, for $l \to \infty$, equals the exact probability that is to be compared with p in (9). Once the accumulated probabilities are calculated, similar inequalities as presented in [15], can be used to decide the validity of $\mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^I \Psi)$ on LJQNs. The accumulated probability is always an underestimation of the actual probability. The value of the maximum error ε accumulated for all states and depending on the number of steps l decreases as l increases. Thus, we obtain the following implications:

$$\text{(a)} \quad \tilde{\pi}^{(l)}(s, Sat(\Psi), t) > p \Rightarrow \pi(s, Sat(\Psi), t) > p$$
$$\text{(b)} \quad \tilde{\pi}^{(l)}(s, Sat(\Psi), t) < p - \varepsilon \Rightarrow \pi(s, Sat(\Psi), t) < p$$

If one of these inequalities (a) or (b) holds, we can decide that the bound $< p$ or $> p$ is met. For the bounds $\leq p$ and $\geq p$, similar implications can be

given. If $\tilde{\pi}(s, Sat(\Psi), t) \in [p, p-\varepsilon]$, we cannot decide whether $\pi(s, Sat(\Psi), t)$ meets the bound p. In this case, increasing l resolves the problem. However, note that in case $p = \pi(s, Sat(\psi), t)$ we cannot decide and iteration does not stop. As already mentioned, for all representative states $\overline{r} \in R(\overline{m} + \overline{l})$ the transient probabilities for all $\overline{s} \in S_{\overline{r}}$ computed with l steps will be the same. Thus, if we can decide whether the bound p is met (case (a) or (b) above), we can be sure that $\mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^{[0,t]}\Psi)$ is independent as of $\overline{m}+\overline{l}$. In that case, we check for all states $\overline{s} \leq \overline{m}+\overline{l}$ whether the accumulated transient probability of reaching a Ψ-state meets the bound p. The states $\overline{s} \in S_b$ that satisfy $\mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^{[0,t]}\Psi)$ form the boundary satisfaction set Sat^{S_b} and the representative states that satisfy $\mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^{[0,t]}\Psi)$ form the representative satisfaction set $Sat^{R(\overline{m}+\overline{l})}(\mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}^{[0,t]}\Psi))$.

For model checking the until operator we need to consider $\prod_{i=1}^{n}(m_i + 2l)$ states in total, depending on the number of steps l considered, the dimension n of the LJQN, and the independence vector $\overline{m}$ of the inner formula. Hence, $\prod_{i=1}^{n}(m_i + 1) - \prod_{i=1}^{n} m_i$ representative states suffice to represent the infinite states space $\mathsf{S} \setminus S_b$.

In [16] we presented a method to efficiently compute the transient probabilities in QBDs and check the until operator with a so-called *dynamic stopping criterion*. This method can also be applied to LJQNs. When iteratively computing the approximation $\tilde{\pi}^{(l)}(s, Sat(\Psi), t)$ and regularly checking whether either (a) or (b) holds for all starting states, the number of considered steps with uniformization is minimized. It is expected that such an approach will lead to similar efficiency improvements.

6 Conclusions

In this paper we presented model checking algorithms for checking CSL properties for a very general class of queueing networks, namely for labeled Jackson queueing networks. The underlying state space of such LJQNs is a highly structured CTMC that is, of infinite size in as many dimensions as there are queues. We introduce a new notion of property independence on LJQNs that is needed for model checking. Steady-state probabilities are computed in (stable) LJQNs with well-known product-form results and transient probabilities are computed with an adaption of our uniformization-based approach. We provided a running example throughout the paper to illustrate our approach.

Note that the model checking procedure for LJQN as introduced in this paper, and the model checking procedure for QBDs [15], in their simplest setting, i.e., the case that the model is a simple M|M|1 queue (which is both a QBD and a LJQN), are in essence the same.

At various points, the presented algorithms can be made more efficient. For instance, applying a dynamic stopping criterion (as in [16]) when model checking the until operator in LJQNs might considerably decrease the number of states that has to be taken into account. To do so, we need to develop efficient data structures to store the step-wise computed probabilities.

The complexity for checking the steady-state operator is linear in the number of queues and the complexity for checking the until operator is given by matrix vector multiplications of matrices that are exponential in the number of queues.

We restricted ourselves to atomic propositions of the form $\bigwedge_{i=1}^{n}(s_i \lesseqgtr m_i)$ for $m_i \in \mathbb{N}$; this facilitates the determination of the independence vector $\overline{m}$. For model checking other properties like the balance of queues or a general threshold, the notion of property independence needs to be (and can be) generalized.

Finally, we developed the model checking algorithm for the until operator with time-bound in $[0, t]$, however, other time intervals can be handled similarly as for finite state CTMCs or QBDs [3,17]. In further work, we will introduce rewards to allow for CSRL model checking. Furthermore, we also consider addressing networks of QBDs. However, to the best of our knowledge, there is no method available to compute steady-state probabilities on such multi-dimensional QBDs, hence, we will have to do without the steady-state operator.

References

1. Abdulla, P., Jonsson, B., Nilsson, M., Saksena, M.: A survey of regular model checking. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 35–48. Springer, Heidelberg (2004)
2. Aziz, A., Sanwal, K., Brayton, R.: Model checking continuous-time Markov chains. ACM Transactions on Computational Logic 1(1), 162–170 (2000)
3. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Model-checking algorithms for continuous-time Markov chains. IEEE Transactions on Software Engineering 29(7), 524–541 (2003)
4. Ballarini, P., Hillston, J.: Compositional csl model checking for boucherie product processes. In: 3rd Workshop on Automated Verfication of Critical Systems (AVOCS'03), DSSE Technical Report DSSE-TR-2003-2 (2003)
5. Brázdil, T., Kucera, A., Strazovský, O.: On the decidability of temporal properties of probabilistic pushdown automata. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 145–157. Springer, Heidelberg (2005)
6. Ciardo, G.: Discrete-time Markovian stochastic Petri nets. In: Computations with Markov Chains, pp. 339–358. Raleigh (1995)
7. Clarke, E., Emerson, E., Sistla, A.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems 8(2), 244–263 (1986)
8. Etessami, K., Yannakakis, M.: Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 340–352. Springer, Heidelberg (2005)
9. Gross, D., Miller, D.R.: The randomization technique as a modeling tool and solution procedure for transient Markov processes. Operations Research 32(2), 343–361 (1984)
10. Harrison, P.G.: Transient behaviour of queueing networks. Journal of Applied Probability 18(2), 482–490 (1981)
11. Jackson, J.R.: Networks of waiting lines. Operations Research 5(4), 518–521 (1957)
12. Jackson, J.R.: Jobshop-like queueing systems. Management Science 10(1), 131–142 (1963)

13. Kleinrock, L.: Queueing Systems. Theory, vol. 1. John Wiley & Sons, Chichester (1975)
14. Matis, T.I., Feldman, R.M.: Transient analysis of state-dependent queueing networks via cumulant functions. Journal of Applied Probability 38(4), 841–859 (2001)
15. Remke, A., Haverkort, B.R., Cloth, L.: Model checking infinite-state markov chains. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 237–252. Springer, Heidelberg (2005)
16. Remke, A., Haverkort, B.R., Cloth, L.: Uniformization with representatives. In: Proc. Workshop on Tools for solving structured Markov chains, ACM press (CD only) (2006)
17. Remke, A., Haverkort, B.R., Cloth, L.: CSL model checking algorithms for QBDs. Theoretical Computer Science (2007) doi: 10.1016/j.tcs.2007.05.007
18. Schnoebelen, P.: The verification of probabilistic lossy channel systems. In: Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P., Siegle, M. (eds.) Validation of Stochastic Systems. LNCS, vol. 2925, pp. 445–465. Springer, Heidelberg (2004)
19. van Moorsel, A.P.A.: Performability Evaluation Concepts and Techniques. PhD thesis, Dept. of Computer Science, University Twente (1993)

Symbolic Simulation-Checking of Dense-Time Automata*

Farn Wang

Dept. of Electrical Engineering
Graduate Institute of Electronic Engineering
National Taiwan University
farn@cc.ee.ntu.edu.tw
http://cc.ee.ntu.edu.tw/~farn,
http://cc.ee.ntu.edu.tw/~farn/red

Abstract. A *model automaton is simulated by* a *specification automaton* if every externally observable transition by the model can also be matched by the specification. In this work, we establish a new formulation of simulation from a model TA to a specification TA. The new formulation allows us to develop a simulation-checking algorithm, in greatest fixpoint style, with zones. We also present a technique to construct an under-approximation of the set of state-pairs to be removed in a fixpoint iteration. The technique does not sacrifice the exactness of our algorithm and could enhance the performance of simulation-checking. Finally, we report the performance of an implementation of our algorithms.

Keywords: simulation, implementation, refinement, equivalence, bisimulation, bisimilarity, dense-time, real-time, embedded, model-checking, timed automata, verification, events.

1 Introduction

In the last two decades, the technology of dense-time system *model-checking* [1] has been well-received by the academia, realized with many tools [7,19,15], and used for the verification of several industrial projects [5]. With model-checking, we represent the *model* as an *automaton* and the *specification* as a temporal logic formula and want to check whether the model satisfies the specification. However, in many applications, engineers may instead envision their *specifications* as automata (state-transition systems). There are two approaches for checking model automata against specification ones. The first is the *language inclusion problem* which checks if all runs of a model are also those of the specification.

* The work is partially supported by NSC, Taiwan, ROC under grants NSC 95-2221-E-002-067 and NSC 95-2221-E-002-072. A complete version of the manuscript with all lemma proofs has been archived by ACM Computing Research Repository (CoRR) with PaperID: cs.LO/0610085 on Oct. 14, 2006.

J.-F. Raskin and P.S. Thiagarajan (Eds.): FORMATS 2007, LNCS 4763, pp. 352–368, 2007.

It was proved in [3] that when both the model and specification are represented as *timed automata (TA)* [3], the language inclusion problem is undecidable. The second is called the *simulation-checking problem* that checks if every externally observable transition that can be made by the model at an instant can also be matched by the specification at the same instant. It was proved that the simulation-checking problem of TAs is in EXPTIME [14]. However, to our knowledge, so far, there has not been an efficient tool that algorithmically checks the simulation between two TAs. In contrast, the model-checking problem of TAs can be efficiently solved with *zones*[1] [7, 11, 15, 19]. In this work, we present a new formulation of simulation between two TAs which enables us to design a simulation-checking algorithm with zones.

We denote the model automaton as A_1 and the specification as A_2. Given a state μ of A_1, a state ν of A_2, and a $\delta \in \mathbb{R}^{\geq 0}$ (the set of non-negative reals), if A_1 can make a transition at δ time units from μ and A_2 cannot match the transition at δ time units from ν, then δ is called a *refuting time* from μ to ν. Intuitively, a *simulation* is a relation between a state μ of A_1 and a state ν of A_2 such that there is no refuting time from μ to ν. A simulation is constructed by starting from an initial image of the simulation and using the *greatest fixpoint* algorithm to iteratively remove state-pairs with a refuting time from the image. After we can remove no more state-pairs, the final image is a simulation. We let C_2^1 be the biggest timing constant used in either A_1 or A_2. Our observation is that in the traditional formulation of simulation between TAs [4, 12, 14], in one iteration, we allow for the removal of state-pairs with any refuting time. But zones do not support the precise recording of any state-pair with unbounded refuting times. With our new formulation, in each iteration, a state-pair is removed if there is a refuting time in $[0, C_2^1]$ for the pair. The removal of state-pairs with the traditional formulation can then be achieved with removals in successive iterations with our new formulation. Based this new formulation, we have developed our algorithm I for the symbolic simulation-checking with zones.

Algorithm I may suffer from low performance since we may need many greatest fixpoint iterations to remove some state-pairs with only large refuting times $> C_2^1$. We thus present algorithm II which uses an under-approximation technique to conservatively remove state-pairs with refuting times $> C_2^1$ in each iteration. The good thing of algorithm II is that it does not sacrifise the exactness of the greatest fixpoint construction and sometimes may run faster. We have implemented our algorithms with TCTL model-checker **RED**, version 7, and report their performance against several benchmarks.

We have the following presentation plan. Section 2 is for related work. Section 3 presents the definition of TAs. Section 4 presents the traditional formulation of simulation between two TAs. Section 5 establishes our new formulation for simulation between TAs and presents algorithm I. Section 6 presents algorithm II. Section 7 reports our implementation and experiment.

[1] A *zone* is a conjunction of atomic propositions and constraints like either $x - y \leq c$ or $x - y < c$ where x, y are either zero or clocks and c is an integer constant.

2 Related Work

In the literature, "*is simulated by*," "*implements*," and "*refines*" are used interchangeably and means the same thing. "*Equivalence*" means "*bisimulation*." Cerans showed that the bisimulation-checking problem of timed processes is decidable [8]. Taşıran et al showed that the simulation-checking problem of TAs is in EXPTIME [14]. They also proposed an algorithm to check whether a location homomorphism between a model TA and a specification TA preserves timed behaviors. The ideas were implemented in Timed COSPAN. However, there is no general strategy to efficiently construct such homomorphisms.

Henzinger et al presented an algorithm that computes the time-abstract simulation that does not preserve timed properties [10].

Nakata also discussed how to do symbolic bisimulation checking with integer-time labeled transition systems [13]. Beyer has implemented a refinement-checking algorithm for TAs with integer-time semantics [6].

Lin and Wang presented a sound proof system for the equivalence of TAs with dense-time semantics [12]. Usually, the proofs may need human guidance.

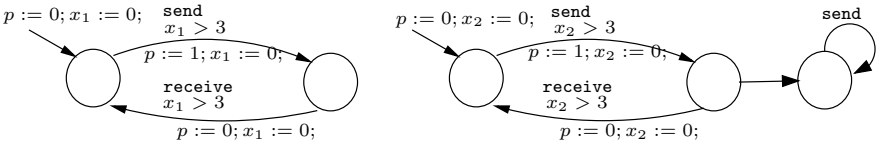
Cleaveland and Steffen proposed to convert the equivalence-checking problem to the model-checking problem [9]. Aceto et al discussed how to construct such a modal logic formula that completely characterizes a TA [4]. However, the formula they constructed is not for TAs with timed invariance constraints. In comparison, our algorithms can handle TAs with timed invariance constraints. Moreover, their characteristic formulas fall in the realm of *linear-hybrid automata (LHA)*. In general, the model-checking problem of LHAs is undecidable [2] and is usually handled with high-complexity manipulation of convex polyhedra [2,16].

3 Timed Automata with Events

Let $\mathbb{N}$ be the set of non-negative integers and $\mathbb{R}^{\geq 0}$ the set of non-negative reals. Also 'iff' means "if and only if." Given a set P of atomic propositions and a set X of clocks, we use $\mathbb{B}(P, X)$ as the set of all Boolean combinations of atoms of the forms p and $x \sim c$, where $p \in P$, $x \in X \cup \{0\}$, '$\sim$' is one of $\leq, <, =, >, \geq$, and c is an integer constant. An element in $\mathbb{B}(P, X)$ is called a *state predicate*.

A *valuation* of a set Y *(domain)* is a mapping from Y to a *codomain*. A *partial valuation* may be undefined for some elements in the domain. If a valuation ν is undefined for y, we denote $\nu(y) = \circledcirc$. When it is not said specifically, a valuation means a *total valuation* that assigns a value to every element in the domain. Given two (partial) valuations ν and ν', $\nu\nu'$ is the composition of ν and ν' defined in the following way: for all y, if $\nu'(y)$ is defined, $\nu\nu'(y) = \nu'(y)$; otherwise $\nu\nu'(y) = \nu(y)$. Given a value v in the domain of a variable x, $[x \leftarrow v]$ denotes the partial mapping that maps x to v and everything else to $\circledcirc$.

Definition 1. Timed automaton with events (TA) A *TA* A is a tuple $\langle \Sigma, X, G, L, I, H, E, \epsilon, \tau, \pi \rangle$ with the following restrictions. Σ is a finite set of event names. X is a finite set of clocks. G is a finite set of global atomic propositions. G and Σ represent the sets of external observables of a TA. L is a finite set



(a) A_1 with $H_1 : \begin{pmatrix} (p = 0 \wedge x_1 < 5) \\ \vee (p = 1 \wedge x_1 < 10) \end{pmatrix}$ (b) A_2 with $H_2 : \begin{pmatrix} (p = 0 \wedge x_2 < 5) \\ \vee (p = 1 \wedge x_2 < 5) \end{pmatrix}$

Fig. 1. Two example TAs

of local atomic propositions such that $G \cap L = \emptyset$. $I \in \mathbb{B}(G \cup L, X)$ is the initial condition. $H \in \mathbb{B}(G \cup L, X)$ is the invariance condition. E is a finite set of transition rules. $\epsilon : E \mapsto 2^{\Sigma}$ labels each rule with a set of events. $\tau : E \mapsto \mathbb{B}(G \cup L, X)$ defines the triggering condition of each rule execution. For each $e \in E$, $\pi(e)$ is a partial valuation from X to $\{0\}$ and from $G \cup L$ to $\{true, false\}$ that defines the assignments to clocks and proposition variables of each rule execution. If $\pi(e)(y)$ is undefined, it means variable y stays unchanged in the transition. For convenience, we assume that for every TA, there is a null transition $\perp$ such that $\epsilon(\perp) = \emptyset$, $\tau(\perp) = true$, and $\pi(\perp)$ is undefined on everything. ■

Example 1. We have the transition diagrams of two example TAs in figure 1. They share events `send` and `receive` and global proposition p. They respectively have local clocks x_1 and x_2. A_1 has two transitions while A_2 has four. We stack the events, triggering conditions, and the assignments made at each transition. The initial conditions are labeled by the arcs without a source. ■

Definition 2. **States of a TA** A *state* of a TA $A = \langle \Sigma, X, G, L, I, H, E, \epsilon, \tau, \pi \rangle$ is a total valuation from X to $\mathbb{R}^{\geq 0}$ and $G \cup L$ to $\{true, false\}$. Let V_A denote the set of states of A. For any state ν and $\delta \in \mathbb{R}^{\geq 0}$, $\nu + \delta$ is a valuation identical to ν except that for every $x \in X$, $(\nu + \delta)(x) = \nu(x) + \delta$. ■

A valuation ν *satisfies* a state-predicate η, in symbols $\nu \models \eta$, if the state-predicate evaluates to true when all its variables are interpreted according to ν. Given two states ν, ν' and a transition e of a TA A, we say A *transits with* e from ν to ν', in symbols $\nu \xrightarrow{e} \nu'$, if $\nu \models \tau(e)$, $\nu\pi(e) = \nu'$, and $\nu' \models H$.

Definition 3. **Runs** Given a TA $A = \langle \Sigma, X, G, L, I, H, E, \epsilon, \tau, \pi \rangle$, a *run* of A is an infinite sequence of state-time pairs $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots\ldots$ with the following three restrictions. (1) $t_0 t_1 \ldots t_k \ldots\ldots$ is a monotonically non-decreasing real-number sequence. (2) For all $k \geq 0$, for all $\delta \in [0, t_{k+1} - t_k]$, $\nu_k + \delta \models H$. (3) For all $k \geq 0$, there is an $e \in E$ such that $\nu_k + t_{k+1} - t_k \xrightarrow{e} \nu_{k+1}$.

A *run segment* is a finite prefix of a run. Given a transition set $E' \subseteq E$, an E'-segment is a run segment $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_n, t_n)$ such that for each $k \in [0, n)$, there is an $e \in E'$ with $\nu_k + t_{k+1} - t_k \xrightarrow{e} \nu_{k+1}$. ■

4 Simulation

Suppose $A_i = \langle \Sigma, X_i, G, L_i, I_i, H_i, E_i, \epsilon_i, \tau_i, \pi_i \rangle$, $1 \leq i \leq 2$. Two transitions $e_1 \in E_1$ and $e_2 \in E_2$ are *compatible* if $\epsilon_1(e_1) = \epsilon_2(e_2)$ and $\forall p \in G(\pi_1(p) = \pi_2(p))$. Given an $e_1 \in E_1$, we let $E_2^{(e_1)} = \{e_2 \mid e_2 \in E_2; e_2 \text{ is compatible with } e_1.\}$.

An $E_2^{(\perp)}$-segment $(\nu_0, t_0) \ldots (\nu_k, t_k)$ of A_2 is called a *pre-matching segment* for μ, e_1, δ, and binary relation $Q \subseteq V_{A_1} \times V_{A_2}$ iff $t_k - t_0 = \delta$, $(\mu + t_k - t_0, \nu) \in Q$, and for every $h \in [0, k)$ and $t' \in [0, t_{h+1} - t_h]$, $(\mu + t_h - t_0 + t', \nu_h + t') \in Q$.

Definition 4. **Simulation** Assume that $A_i = \langle \Sigma, X_i, G, L_i, I_i, H_i, E_i, \epsilon_i, \tau_i, \pi_i \rangle$, $1 \leq i \leq 2$, $L_1 \cap L_2 = \emptyset$, and $X_1 \cap X_2 = \emptyset$. A *simulation* from A_1 to A_2 is a binary relation $Q \subseteq V_{A_1} \times V_{A_2}$ with the following two restrictions on all its element (μ_0, ν_0). (1) μ_0 and ν_0 agree on valuation of variables in G. (2) For every $\delta \in \mathbb{R}^{\geq 0}$ and transition e_1 of E_1, if $\mu_0 + \delta \models \tau_1(e_1)$, $(\mu_0 + \delta)\pi_1(e_1) \models H_1$, and $\forall \hat{\delta} \in [0, \delta](\mu_0 + \hat{\delta} \models H_1)$ are all true, then there are an $e_2 \in E_2^{(e_1)}$ and a pre-matching segment $(\nu_0, t_0) \ldots (\nu_n, t_n)$ of A_2, for μ, e_1, δ, and Q such that $\nu_n \models \tau_2(e_2)$, $\nu_n \pi_2(e_2) \models H_2$, and $((\mu_0 + t_n - t_0)\pi_1(e_1), \nu_n \pi_2(e_2)) \in Q$.

Given a simulation Q from A_1 to A_2, we denote $A_1 \preceq_Q A_2$ if for every state $\mu \models I_1$, there is a $\nu \models I_2$ with $(\mu, \nu) \in Q$. If $\exists Q(A_1 \preceq_Q A_2)$, we say A_1 *is simulated by* A_2, in symbols $A_1 \preceq A_2$. ■

Note that we require a pair (μ, ν) in a simulation must agree on the valuations of variables in G. Thus for all $g \in G$, $\mu(g) = \nu(g) = \mu\nu(g)$. With this observation, from now on, we represent (μ, ν) as valuation $\mu\nu$. With this representation, we can conveniently reason with constraints on pairs in a simulation.

There can be many simulations between two TAs. In general, we are interested at the *maximal* simulation[2] which properly contains other simulations. From now on, we assume that the simulations in discussion are always maximal.

Example 2. In example 1, A_1 is not simulated by A_2 since A_1 can make a transition after 5 time units in state $q = 1$ while A_2 cannot. ■

In the following, we discuss the traditional formulation of simulation between TAs [14]. Given an $e_1 \in E_1$ and a $\mu \in V_{A_1}$, we let $play_{e_1}^{\delta}(\mu)$ denote the condition that A_1 can do transition e_1 at time δ from state μ. Formally speaking,

$$play_{e_1}^{\delta}(\mu) \stackrel{\text{def}}{=} \mu + \delta \models \tau_1(e_1) \wedge (\mu + \delta)\pi_1(e_1) \models H_1 \wedge \neg \exists 0 \leq \delta' \leq \delta (\mu + \delta' \models \neg H_1)$$

Given a $\mu \in V_{A_1}$, a ν, a $\nu' \in V_{A_2}$, a $\delta \in \mathbb{R}^{\geq 0}$, and a $Q \subseteq V_{A_1} \times V_{A_2}$, we let $pre\text{-}match_2^{\delta}(\mu, \nu, \nu', Q)$ denote the condition that there exists a pre-matching segment $(\nu_0, t_0) \ldots (\nu_k, t_k)$ such that $\nu_0 = \nu$, $\nu_k = \nu'$, and $t_k - t_0 = \delta$. Given an $e_1 \in E_1$, a state-pair $\mu\nu \in V_{A_1} \times V_{A_2}$, and a state-pair subset $Q \subseteq V_{A_1} \times V_{A_2}$, we let $match_{e_1}^{\delta}(\mu, \nu, Q)$ denote the condition that A_2 can match e_1 at time δ from state ν with respect to Q. Note that $match_{e_1}^{\delta}(\mu, \nu, Q)$ is a predicate on state-pairs and values of δ. Formally speaking,

[2] We assume that for every state-pair $\mu\nu$ in the maximal simulation, $\mu \models H_1$ and $\nu \models H_2$.

$$match^{\delta}_{e_1}(\mu,\nu,Q) \stackrel{\text{def}}{=} \exists \nu' \in V_{A_2} \left(\begin{array}{l} pre\text{-}match^{\delta}_2(\mu,\nu,\nu',Q) \\ \wedge \exists e_2 \in E_2^{(e_1)} \left(\begin{array}{l} \nu' \models \tau_2(e_2) \wedge \nu'\pi_2(e_2) \models H_2 \\ \wedge((\mu+\delta)\pi_1(e_1))(\nu'\pi_2(e_2)) \in Q \end{array} \right) \end{array} \right)$$

Note that if $play^{\delta}_{e_1}(\mu) \wedge \neg match^{\delta}_{e_1}(\mu,\nu,Q)$ is true, then δ is a *refuting time* from μ to ν in Q. Given an interval D with lower-bound and upper-bound in $\mathbb{N}$, we let $not\_sim^{D}_{e_1}(\mu,\nu,Q) \stackrel{\text{def}}{=} \exists \delta \in D(play^{\delta}_{e_1}(\mu) \wedge \neg match^{\delta}_{e_1}(\mu,\nu,Q))$. A simulation can be concisely reformulated with the following lemma.

Lemma 1. *Assume that* $A_i = \langle \Sigma, X_i, G, L_i, I_i, H_i, E_i, \epsilon_i, \tau_i, \pi_i \rangle$, $1 \leq i \leq 2$, $L_1 \cap L_2 = \emptyset$, *and* $X_1 \cap X_2 = \emptyset$. Q *is a simulation from* A_1 *to* A_2 *iff* $Q = \{\mu\nu \mid \mu \models H_1, \nu \models H_2, \neg\exists e_1 \in E_1(\text{not\_sim}^{\mathbb{R}^{\geq 0}}_{e_1}(\mu,\nu,Q))\}$. ■

Lemma 1 implies that we can implement a simulation-checking procedure in the traditional greatest fixpoint style. That is, we first let $Q := V_{A_1} \times V_{A_2}$; and then iteratively remove set $\{\mu\nu \mid \exists e_1 \in E_1(not\_sim^{\mathbb{R}^{\geq 0}}_{e_1}(\mu,\nu,Q))\}$ from Q until a fixpoint is reached.

5 Algorithm I for Simulation Checking

We plan to present the section as follows. Subsection 5.1 reformulates simulation between TAs. Subsection 5.2 discusses how to use zones to characterize state-pairs and δ values. Subsection 5.3 explains the basic procedures that we need. Subsections 5.4 and 5.5 present predicates respectively for $play^{\delta}_{e_1}(\mu)$ and $match^{\delta}_{e_1}(\mu,\nu,Q)$ with $\delta \in [0, C^1_2]$. Subsection 5.6 constructs a predicate for our new formulation and presents our first algorithm for simulation-checking.

5.1 A New Formulation for Simulation

Suppose we are given an $e_1 \in E_1$, a $\delta \in \mathbb{R}^{\geq 0}$, a run $\rho = (\nu_0,t_0)\ldots(\nu_k,t_k)\ldots$ of A_2, a $k \geq 0$, and a $t \in [0, t_{k+1} - t_k]$. We say $t_k - t_0 + t$ is a *disrupting time* in ρ for μ, e_1, δ, and Q iff $t_k - t_0 + t < \delta$ and $(\mu + t_k + t - t_0)(\nu_k + t) \notin Q$. A run with a disrupting time for μ, e_1, δ, and Q is called a *disrupted run* for μ, e_1, δ, and Q. We define $minT^{\delta}_{e_1}(\mu,\rho,Q)$ as the smallest disrupting time in ρ for μ, e_1, δ, and Q. We let $maxminT^{\delta}_{e_1}(\mu,\nu,Q)$ be defined as follows.

$$maxminT^{\delta}_{e_1}(\mu,\nu,Q) = \max \left\{ minT^{\delta}_{e_1}(\mu,\rho,Q) \,\middle|\, \begin{array}{l} \rho \text{ is a disrupted run from } \nu \\ \text{of } A_2 \text{ for } \mu, e_1, \delta, \text{ and } Q \end{array} \right\}.$$

If there is neither a disrupting time $< \delta$ in ρ nor a state at δ time units from the starting state of ρ to execute a transition matching e_1, then we call ρ a *futile run* for μ, e_1, δ, and Q. It is easy to see that if given a $\mu\nu \in Q$ such that $play^{\delta}_{e_1}(\mu) \wedge \neg match^{\delta}_{e_1}(\mu,\nu,Q)$ is true, then we know all runs of A_2 starting from ν must be either disrupted or futile for μ, e_1, δ, and Q. Here we have the following central lemma of the work.

Lemma 2. **[Central Lemma]** *Suppose we are given a* $Q \subseteq V_{A_1} \times V_{A_2}$, *an* $e_1 \in E_1$, *and a* $\delta \in \mathbb{R}^{\geq 0}$. *If there is a* $\mu\nu \in Q$ *such that* $\text{play}^{\delta}_{e_1}(\mu) \wedge \neg\text{match}^{\delta}_{e_1}(\mu,\nu,Q)$

is true, then for each $t \in [0, \delta]$*, there is a* $\bar{\mu}\bar{\nu} \in Q$ *such that either* $\text{play}^t_{e_1}(\bar{\mu}) \wedge \neg\text{match}^t_{e_1}(\bar{\mu}, \bar{\nu}, Q)$ *or* $\text{play}^t_{\perp}(\bar{\mu}) \wedge \neg\text{match}^t_{\perp}(\bar{\mu}, \bar{\nu}, Q)$ *is true.*

Proof : There are two cases to analyze. In the first case, we assume that there is a futile run $\rho = (\nu_0, t_0) \ldots (\nu_k, t_k) \ldots$ for μ, e_1, δ, and Q. Then there is a $k \geq 0$ and a $t' \in [0, t_{k+1} - t_k]$ such that $\delta - (t_k + t' - t_0) = t$ in ρ. Since all runs from ν are either disrupted or futile for μ, e_1, δ, and Q, we can prove that all runs from $\nu_k + t'$ are also either disrupted or futile for μ, e_1, t, and Q. Thus $\mu + \delta - t$ is a candidate for $\bar{\mu}$ while $\nu_k + t'$ is for $\bar{\nu}$. Thus this case is proven.

In the second case, we assume that all runs of A_2 from ν are disrupted for μ, e_1, δ, and Q. If $t > \mathit{maxmin}T^{\delta}_{e_1}(\mu, \nu, Q)$, according to the definition of simulation, $\mathit{play}^t_{\perp}(\mu) \wedge \neg \mathit{match}^t_{\perp}(\mu, \nu, Q)$ is also true. Then the lemma is proven with $\mu = \bar{\mu}$ and $\nu = \bar{\nu}$. If $t \leq \mathit{min}T^{\delta}_{e_1}(\mu, \rho, Q) = \mathit{maxmin}T^{\delta}_{e_1}(\mu, \nu, Q)$, we let $\rho = (\nu_0, t_0) \ldots (\nu_k, t_k) \ldots$ of A_2 be a disrupted run for μ, e_1, δ, and Q such that $\nu_0 = \nu$ and $\mathit{min}T^{\delta}_{e_1}(\mu, \rho, Q) = \mathit{maxmin}T^{\delta}_{e_1}(\mu, \nu, Q)$. Let $\nu_k + t'$ with $t' \in [0, t_{k+1} - t_k]$ be the state in ρ that causes the disrupting. Since time is continuous, we can find an $h \in [0, k]$ and a $t'' \in [0, t_{h+1} - t_h]$ such that $t_h - t_0 + t'' + t \geq t_k - t_0 + t'$. Then with an argument similar to the one in the first case, we can prove the case with $\bar{\mu} = \mu + t_h - t_0 + t''$ and $\bar{\nu} = \nu_h + t''$. ■

Lemma 2 implies the following lemma for a new formulation of simulation.

Lemma 3. *Suppose that* $A_i = \langle \Sigma, X_i, G, L_i, I_i, H_i, E_i, \epsilon_i, \tau_i, \pi_i \rangle$*,* $1 \leq i \leq 2$*,* $L_1 \cap L_2 = \emptyset$*, and* $X_1 \cap X_2 = \emptyset$*. Q is a simulation from* A_1 *to* A_2 *iff* $Q = \{\mu\nu \mid \mu \models H_1, \nu \models H_2, \neg\exists e_1 \in E_1 \exists \delta \in [0, C^1_2](\text{play}^{\delta}_{e_1}(\mu) \wedge \neg\text{match}^{\delta}_{e_1}(\mu, \nu, Q))\}$. ■

5.2 Representation with Zones

A *zone-predicate* ζ of a set P of atomic propositions, a set X of clocks, and a $c \in \mathbb{N}$ is constructed inductively with the following rules.

$$\zeta ::= p \mid x - y \sim d \mid \neg\zeta_1 \mid \zeta_1 \wedge \zeta_2$$

Here $p \in P$, $x, y \in X \cup \{0\}$, '$\sim$' $\in \{\leq, <, =, \neq, >, \geq\}$, and d is an integer in $[-c, c]$. Let $Z_c(P, X)$ be the set of zone-predicates that can be constructed of P, X, and c. A zone-predicate is *conjunctive* if the only Boolean operator it can have is $\wedge$. A state-space that can be characterized with a conjunctive zone-predicate is called a *zone*. A *region* of P, X, and c is a smallest zone that cannot properly contain other non-empty zones characterizable in $Z_c(P, X)$. A valuation satisfies a zone-predicate if the zone-predicate is evaluated true when all its variables are interpreted according to the valuation.

Given two state-pairs $\mu\nu$ and $\mu'\nu'$ in $V_{A_1} \times V_{A_2}$, we denote $\mu\nu \equiv \mu'\nu'$ iff $\mu\nu$ and $\mu'\nu'$ are in the same region in $Z_{C^1_2}(G \cup L_1 \cup L_2, X_1 \cup X_2)$. Here we restate theorem 10 about simulation-checking with zones in [14].

Theorem 1. *Given the maximal simulation* Q *from* A_1 *to* A_2 *and two state-pairs* $\mu\nu \equiv \mu'\nu'$ *in* $V_{A_1} \times V_{A_2}$*,* $\mu\nu \in Q$ *iff* $\mu'\nu' \in Q$*.* ■

Lemma 3 and theorem 1 motivate us to represent $not\_sim_{e_1}^{[0,C_2^1]}(\mu,\nu,Q)\}$ with zones. In the formulation of $not\_sim_{e_1}^{D}(\mu,\nu,Q)$, we use a variable δ to measure the value of refuting times. One difficulty to use zones to represent this variable δ is that its values decrement with time progress while the values of clocks increment. The increment rates of clocks are uniform and exactly the same as the decrement rate of δ. To overcome this difficulty, we introduce an auxiliary clock 'Ω^δ' for the description of constraints on δ. The value of clock Ω^δ always equals $C_2^1-\delta$. Its values are in $(-\infty, C_2^1]$ with the same increment rate as other clocks. A valuation (state-pair or state) with $\Omega^\delta = C_2^1$ can be at the moment that a transition is made by A_1 and a matching transition from A_2 is expected. In this section, we only consider those valuations γ with $\gamma(\Omega^\delta) \in [0, C_2^1]$. Clock Ω^δ is neither used nor initialized in A_1 and A_2. Given two valuations μ and ν, if $\mu(\Omega^\delta) = \nu(\Omega^\delta)$, then we know they have the same timing distance to the moment when A_1 makes transition e_1 and A_2 is expected to match.

Example 3. Given $G=\{a\}, X_1=\{x_1\}, L_1=\{b_1\}, X_2=\{x_2\}, L_2=\{b_2\}$, we may have the following zone-predicate for a fixpoint image.

$$\mathcal{Q} \equiv \begin{array}{l}(a \wedge b_1 \wedge \neg b_2 \wedge 0 \le x_1 \wedge 3 < x_2 \le 5 \wedge x_2 - x_1 \le 5 \wedge \Omega^\delta \le 5)\\ \vee\ (\neg a \wedge 2 \le x_1 < 9 \wedge 1 < x_2 \wedge x_1 - x_2 < 8 \wedge -\Omega^\delta \le -1 \wedge \Omega^\delta \le 2)\end{array}$$ ■

5.3 Basic Procedures

The *existential quantification* of a formula η on a variable y is $\exists y(\eta)$. For zone-predicates, efficient implementation has been discussed in [11,15,16]. We use $\mathrm{EXQ}(\eta, X')$ to denote an implementation of the existential quantification that eliminates clock variables in X' from η.

Example 4. For the $\mathcal{Q}$ in example 3,

$$\exists a(\mathcal{Q}) \equiv \begin{array}{l}(b_1 \wedge \neg b_2 \wedge 0 \le x_1 \wedge 3 < x_2 \le 5 \wedge x_2 - x_1 \le 5)\\ \vee\ (2 \le x_1 < 9 \wedge 1 < x_2 \wedge x_1 - x_2 < 8)\end{array}$$

and $\exists x_1(\mathcal{Q}) \equiv (a \wedge b_1 \wedge \neg b_2 \wedge 3 < x_2 \le 5) \vee (\neg a \wedge 1 < x_2 < 6)$. ■

Given two zone-predicates ζ, ζ', $\mathtt{Tbck}(\zeta', \zeta)$ computes the precondition of ζ through a time-progress through states that satisfy ζ'. According to [11], we have $\mathtt{Tbck}(\zeta_1,\zeta_2) \stackrel{\text{def}}{=} \exists t\,(t \ge 0 \wedge \zeta_2 + t \wedge \neg\exists t'((\neg\zeta_1) + t' \wedge 0 \le t' \wedge t' \le t))$.

Given a partial assignment Π of $G \cup L_1 \cup L_2 \cup X_1 \cup X_2$, we let $\eta\Pi$ be the precondition to η before the assignment. If Π is defined on $y_1, \ldots, y_n$, then $\eta\Pi \stackrel{\text{def}}{=} (\bigwedge_{x \text{ is a clock defined in } \Pi.} x \ge 0) \wedge \exists y_1 \ldots \exists y_n(\eta \wedge \bigwedge_{1\le i\le n} y_i = \Pi(y_i))$. Given an $e_1 \in E_1$ and an $e_2 \in E_2^{(e_1)}$, the weakest precondition to η through discrete transition pair (e_1, e_2) can be represented as

$$\mathtt{Xbck}_{(e_1,e_2)}(\eta) \stackrel{\text{def}}{=} \tau_1(e_1) \wedge \tau_2(e_2) \wedge (\eta(\pi_1(e_1)\pi_2(e_2))).$$

5.4 Construction of $\texttt{Play}_{e_1}^{[0,C_2^1]}$ with Zones

We let $\texttt{Play}_{e_1}^{[0,C_2^1]} \stackrel{\text{def}}{=} -\Omega^\delta \le 0 \wedge \Omega^\delta \le C_2^1 \wedge \texttt{Tbck}\left(H_1, \texttt{Xbck}_{(e_1,\perp)}(H_1) \wedge \Omega^\delta = C_2^1\right)$. Here $\texttt{Tbck}\left(H_1, \texttt{Xbck}_{(e_1,\perp)}(H_1) \wedge \Omega^\delta = C_2^1\right)$ is the standard representation of the weakest timed precondition with transition $(e_1, \perp)$ [11,15]. The following lemma shows the correctness of $\texttt{Play}_{e_1}^{[0,C_2^1]}$.

Lemma 4. *Given a state $\mu \in V_{A_1}$ and a real number $t \in [0, C_2^1]$, $\text{play}_{e_1}^t(\mu)$ iff $\mu[\Omega^\delta \leftarrow C_2^1 - t] \models \texttt{Play}_{e_1}^{[0,C_2^1]}$.* ■

5.5 Construction of $\texttt{Match}_{e_1}^{[0,C_2^1]}(Q)$ with Zones

With procedures $\texttt{Xbck}_{(e_1,e_2)}()$, $\texttt{Tbck}()$, and symbolic characterizations for ν' and Q, a symbolic characterization of *pre-match*$_2^t(\mu, \nu, \nu', Q)$, denoted $\texttt{Rbck}_{E_2^{(\perp)}}^{[0,C_2^1]}(\eta_1, \eta_2)$, can be constructed with backward reachability analysis [11,15] within C_2^1 time units. Again, we use clock Ω^δ to measure the length of the reachability run-segments. Computationally, $\texttt{Rbck}_{E_2^{(\perp)}}^{[0,C_2^1]}(\eta_1, \eta_2)$ is the following least fixpoint:

$$\textbf{lfp}Y.\left(\Omega^\delta = C_2^1 \wedge \eta_2 \vee \texttt{Tbck}\left(\begin{pmatrix}\Omega^\delta \le C_2^1 \\ \wedge 0 \le \Omega^\delta \wedge \eta_1\end{pmatrix}, \bigvee_{e_2 \in E_2^{(\perp)}} \texttt{Xbck}_{(\perp,e_2)}(Y)\right)\right).$$

We have the following lemma.

Lemma 5. *Given a zone-predicate $\mathcal{Q}$ for a $Q \subseteq V_{A_1} \times V_{A_2}$, a $\mu\nu \in Q$, a zone-predicate $f_{\nu'}$ for states $\nu' \in V_{A_2}$, and a $t \in [0, C_2^1]$, $\text{pre-match}_2^t(\mu, \nu, \nu', Q)$ is true iff $\mu\nu[\Omega^\delta \leftarrow C_2^1 - t] \models \texttt{Rbck}_{E_2^{(\perp)}}^{[0,C_2^1]}(\mathcal{Q}, f_{\nu'})$.* ■

Specifically, the quantified ν' is for the precondition of all transitions in E_2 that match e_1. Given a zone-predicate $\mathcal{Q}$, the zone-predicate for ν', i.e. $f_{\nu'}$, is $\bigvee_{e_2 \in E_2^{(e_1)}} \texttt{Xbck}_{(e_1,e_2)}(\mathcal{Q})$. Note that to make sure that A_1 and A_2 observe the same behavior with e_1 and e_2 respectively, we construct the precondition of both e_1 and e_2 out of $\mathcal{Q}$. Now with the formulation of $f_{\nu_2'}$, we let

$$\texttt{Match}_{e_1}^{[0,C_2^1]}(\mathcal{Q}) \stackrel{\text{def}}{=} \texttt{Rbck}_{E_2^{(\perp)}}^{[0,C_2^1]}\left(\mathcal{Q}, \bigvee_{e_2 \in E_2^{(e_1)}} \texttt{Xbck}_{(e_1,e_2)}(\mathcal{Q})\right).$$

We can establish the following lemma.

Lemma 6. *Given a zone-predicate $\mathcal{Q}$ for a $Q \subseteq V_{A_1} \times V_{A_2}$, a $\mu\nu \in Q$, and a $t \in [0, C_2^1]$, $\text{match}_{e_1}^t(\mu, \nu, Q)$ is true iff $\mu\nu[\Omega^\delta \leftarrow C_2^1 - t] \models \texttt{Match}_{e_1}^{[0,C_2^1]}(\mathcal{Q})$.* ■

5.6 Algorithm I

Given a zone-predicate $\mathcal{Q}$ for a $Q \subseteq V_{A_1} \times V_{A_2}$, we have the following shorthand.

$$\texttt{Not\_sim}_{e_1}^{[0,C_2^1]}(\mathcal{Q}) \stackrel{\text{def}}{=} \text{EXQ}\left(\texttt{Play}_{e_1}^{[0,C_2^1]} \wedge \neg\texttt{Match}_{e_1}^{[0,C_2^1]}(\mathcal{Q}), \{\Omega^\delta\}\right).$$

Based on lemmas 4 and 6, we can establish the following lemma.

Lemma 7. *Given a zone-predicate $\mathcal{Q}$ for a $Q \subseteq V_{A_1} \times V_{A_2}$, an $e_1 \in E_1$, and a $\mu\nu \in S$,* $\mathrm{not\_sim}_{e_1}^{[0,C_2^1]}(\mu, \nu, Q)$ *is true iff* $\mu\nu \models \mathtt{Not\_sim}_{e_1}^{[0,C_2^1]}(\mathcal{Q})$. ■

With lemma 7 and theorem 1, we now present our algorithm for simulation checking. We start from $\mathcal{Q} = H_1 \wedge H_2$. Then we iteratively delete zones in $\mathtt{Not\_sim}_{e_1}^{[0,C_2^1]}(\mathcal{Q})$ for each $e_1 \in E_1$ from $\mathcal{Q}$ until a fixpoint is reached.

$\mathrm{Sim\_Check}^{[0,C_2^1]}(A_1, A_2)$ /* $A_i = \langle \Sigma, X_i, G, L_i, I_i, H_i, E_i, \epsilon_i, \tau_i, \pi_i \rangle$, $1 \leq i \leq 2$ */ {
  let $\mathcal{Q} := H_1 \wedge H_2$; $\mathcal{Q}' := \mathit{false}$;
  while $(\mathcal{Q} \neq \mathcal{Q}')$, do {
    $\mathcal{Q}' := \mathcal{Q}$;
    for each $e_1 \in E_1$,
      $\mathcal{Q} := \mathcal{Q} \wedge \neg \mathtt{Not\_sim}_{e_1}^{[0,C_2^1]}(Q)$; ..... (A)
    if $(I_1 \neq \mathrm{EXQ}(I_1 \wedge I_2 \wedge \mathcal{Q}, L_2 \cup X_2))$ ..... (B)
      print "A_1 does not implement A_2." and return *false*.
  }
  print "A_1 implements A_2." and return *true*.
}

In statement (B), we check whether all initial states of A_1 is paired with some initial states of A_2 in $\mathcal{Q}$. The statement employs a technique called *early decision of greatest fixpoint (EDGF)* [18] and could significantly reduce the computation time of greatest fixpoint evaluation when no simulation exists. The following lemma establishes the correctness of our algorithm.

Lemma 8. *Sim_Check*$^{[0,C_2^1]}(A_1, A_2)$ *halts. It returns true iff* $A_1 \preceq A_2$.
Proof sketch: It halts since the number of zones (up to timing constant C_2^1) is finite and the while-loop iteratively removes zones from the greatest fixpoint image. The second sentence is true with the following reasons. Theorem 1 and lemma 7 show the correctness of the implementation of $\mathit{not\_sim}_{e_1}^{[0,C_2^1]}(\mu, \nu, Q)$ for $\mu\nu \models \mathtt{Not\_sim}_{e_1}^{[0,C_2^1]}(\mathcal{Q})$. At each iteration of the while-loop, we remove some zones violating the simulation from the greatest fixpoint image. Thus it is clear that when the while-loop halts, there is no zones in the image violating the simulation and we should return *true*. When we find some initial states of A_1 are not in the image at statement (B), we should return *false* since the initial states will not be in the image which is non-increasing in the iterations. ■

6 Algorithm II with an Under-Approximation of $\mathit{not\_sim}_{e_1}^{(C_2^1,\infty)}(Q)$

Assume that we have a set $Q \subseteq V_{A_1} \times V_{A_2}$ at an iteration of the greatest fixpoint loop in procedure $\mathrm{Sim\_Check}^{[0,C_2^1]}(A_1, A_2)$. If this is not the last iteration, some state-pairs will be eliminated from $\mathcal{Q}$ in this iteration. To make sure that the result greatest fixpoint is maximal, it is curcial that no state-pairs in the maximal

greatest fixpont are eliminated from $\mathcal{Q}$. This implies that if we only eliminate an under-approximation of $not\_sim_{e_1}^{(C_2^1,\infty)}(Q)$ in each iteration, then we not only maintain the correctness of the greatest fixpoint calculation but also may have a faster greatest fixpoint calculation. Thus, we propose the following under-approximation of $not\_sim_{e_1}^{(C_2^1,\infty)}(\mu,\nu,Q)$.

$$\begin{aligned}
&under_not_sim_{e_1}^{(C_2^1,\infty)}(\mu,\nu,Q)\\
\stackrel{\text{def}}{=}&\mu\nu\in Q\wedge\forall\delta\in(C_2^1,\infty)(play_{e_1}^{\delta}(\mu)\wedge\neg match_{e_1}^{\delta}(\mu,\nu,Q))\\
\equiv&\mu\nu\in Q\wedge(\forall\delta\in(C_2^1,\infty)(play_{e_1}^{\delta}(\mu)))\wedge\neg\exists\delta\in(C_2^1,\infty)(match_{e_1}^{\delta}(\mu,\nu,Q))
\end{aligned}$$

The following lemma shows why it is an under-approximation.

Lemma 9. *Given a* $Q\subseteq V_{A_1}\times V_{A_2}$, *a* $e_1\in E_1$, *and a* $\mu\nu\in Q$, *if* $\text{under\_not\_sim}_{e_1}^{(C_2^1,\infty)}(\mu,\nu,Q)$ *is true, then* $\text{not\_sim}_{e_1}^{(C_2^1,\infty)}(\mu,\nu,Q)$ *is true.* ■

In the following, we present techniques for the construction of characterizations for the components in this under-approximation.

6.1 Construction of $\texttt{UPlay}_{e_1}^{(C_2^1,\infty)}(Q)$ with Zones

The following lemma helps us to characterize $play_{e_1}^{\delta}(\mu)$ with zone-predicates.

Lemma 10. *Given a state* $\mu\in V_{A_1}$ *and a* $t\in(C_2^1,\infty)$, $\text{play}_{e_1}^{t}(\mu)$ *is true iff for every* $t'\in(C_2^1,\infty)$, $\text{play}_{e_1}^{t'}(\mu)$ *is true.*

Proof : Zone-predicates are with inequalities like $x-x'\sim d$ where $|d|\leq C_2^1$. Thus for any $t,t'\in(C_2^1,\infty)$, $\mu+t\models x-x'\sim d$ iff $\mu+t'\models x-x'\sim d$. ■

Since (C_2^1,∞) is non-empty, lemma 10 implies that for any $\mu\in V_{A_1}$, the truth values of $\forall\delta\in(C_2^1,\infty)(play_{e_1}^{\delta}(\mu))$ and $\exists\delta\in(C_2^1,\infty)(play_{e_1}^{\delta}(\mu))$ are the same. We let $\texttt{UPlay}_{e_1}^{(C_2^1,\infty)}$ denote the following zone-predicate.

$$\texttt{UPlay}_{e_1}^{(C_2^1,\infty)}\stackrel{\text{def}}{=}\text{EXQ}\left(\Omega^{\delta}<0\wedge\texttt{Tbck}\left(H_1,\texttt{Xbck}_{(e_1,\perp)}(H_1)\wedge\Omega^{\delta}=C_2^1\right),\{\Omega^{\delta}\}\right)$$

Then we have the following lemma.

Lemma 11. *Given a* $\mu\in V_{A_1}$, $\forall\delta\in(C_2^1,\infty)\left(\text{play}_{e_1}^{\delta}(\mu)\right)$ *is true iff* $\mu\models\texttt{UPlay}_{e_1}^{(C_2^1,\infty)}$. ■

6.2 Construction of $\texttt{OMatch}_{e_1}^{(C_2^1,\infty)}(Q)$ with Zones

There are several observations that we can make for the evaluation of $\exists\delta\in(C_2^1,\infty)(match_{e_1}^{\delta}(\mu,\nu,Q))$. First, for a state-pair $\mu\nu$ such that $t>C_2^1$ and $match_{e_1}^{t}(\mu,\nu,Q)$ is true, then for all state-pairs $\mu'\nu'$ that can forwardly reach $\mu\nu$ through a pre-matching segment for μ', e_1, t', and Q, we also know $t'+t\geq t>C_2^1$. Second, according to the discussion in the previous section, for those state-pairs $\mu\nu$ with $\delta\leq C_2^1$ and $match_{e_1}^{\delta}(\mu,\nu,Q)$ true for some Q, we can use zone-predicates

to characterize the relation between $\mu\nu$, δ, and Q for the calculation of backward reachability. We use the following two zone-predicate variables to record the information we need for the evaluation of $\exists\delta \in (C_2^1, \infty)(match^{\delta}_{e_1}(\mu, \nu, Q))$.

Π : for the recording of those state-pairs $\mu\nu$ and time $t \leq C_2^1$ such that there is a pre-matching segment of t time units from $\mu\nu[\Omega^{\delta} \leftarrow C_2^1 - t]$ to some $\bar{\mu}\bar{\nu}[\Omega^{\delta} \leftarrow C_2^1]$ in $\Omega^{\delta} = C_2^1 \wedge \bigvee_{e_2 \in E_2^{(e_1)}} \mathtt{Xbck}_{(e_1,e_2)}(\mathcal{Q})$.

Δ : for the recording of those state-pairs $\mu\nu$ which are backwardly reachable from a state-pair in $\Omega^{\delta} = C_2^1 \wedge \bigvee_{e_2 \in E_2^{(e_1)}} \mathtt{Xbck}_{(e_1,e_2)}(\mathcal{Q})$ through a finite run segment of length $> C_2^1$.

The precondition of state-pairs in Δ will also be characterized in Δ eventually. The precondition of state-pairs in Π may end up in either Π or Δ. Moreover, the precondition calculation of Δ does not involve variable δ and clock Ω^{δ} in this approximation. With these observations, we present the following algorithm that works on Δ and Π to calculate $\exists\delta \in (C_2^1, \infty)(match^{\delta}_{e_1}(\mu, \nu, Q))$.

$\mathtt{OMatch}^{(C_2^1,\infty)}_{e_1}(\mathcal{Q})$ {
  $\Pi := \Omega^{\delta} = C_2^1 \wedge \bigvee_{e_2 \in E_2^{(e_1)}} \mathtt{Xbck}_{(e_1,e_2)}(\mathcal{Q})$;
  $\Delta := \mathit{false}$; $\Pi' := \mathit{false}$; $\Delta' := \mathit{false}$; ..... (C)
  while $\Pi \neq \Pi' \vee \Delta \neq \Delta'$, do {
    $\Delta' := \Delta$; $\Pi' := \Pi$;
    $\Delta := \Delta \vee \mathtt{Tbck}(\Delta, \mathtt{Xbck}_{(e_1,\perp)}(\Delta))$; ..... (D)
    $\Gamma := \mathtt{Tbck}(\mathcal{Q}, \mathtt{Xbck}_{(e_1,\perp)}(\Pi))$; $\Pi := \Pi \vee (\Omega^{\delta} \geq 0 \wedge \Gamma)$; ..... (E)
    $\Delta := \Delta \vee \mathrm{EXQ}(\Omega^{\delta} < 0 \wedge \Gamma, \{\Omega^{\delta}\})$; ..... (F)
  }
  return Δ; ..... (G)
}

Note that the only changes that we can make to Π and Δ are at statements (D), (E), and (F). Every time when we change Π at statement (E), we make sure that the δ values in Π are in $[0, C_2^1]$ through conjunction with constraint $\Omega^{\delta} \geq 0$. Statement (D) calculates the timed weakest precondition from all state-pairs already in Δ with δ values already in (C_2^1, ∞). Thus it can only add predicates with δ values in (C_2^1, ∞). When statement (F) calculates the timed weakest precondition from all state-pairs already in Π, we make sure that all δ values added to Δ are in (C_2^1, ∞) through conjunction with constraint $\Omega^{\delta} < 0$.

The following lemma shows the correctness of $\mathtt{OMatch}^{(C_2^1,\infty)}_{e_1}(\mathcal{Q})$.

Lemma 12. *At statement (G) of procedure* $\mathtt{OMatch}^{(C_2^1,\infty)}_{e_1}(\mathcal{Q})$*, for all* $\mu\nu \in V_{A_1} \times V_{A_2}$*,* $\mu\nu \models \Delta$ *iff* $\mu\nu \models \exists\delta \in (C_2^1, \infty)(\mathrm{match}^{\delta}_{e_1}(\mu, \nu, Q))$. ■

6.3 Algorithm II

Given a zone-predicate $\mathcal{Q}$, we define

$$\mathtt{UNot\_sim}^{(C_2^1,\infty)}_{e_1}(\mathcal{Q}) \stackrel{\mathrm{def}}{=} \mathtt{UPlay}^{(C_2^1,\infty)}_{e_1} \wedge \neg\mathtt{OMatch}^{(C_2^1,\infty)}_{e_1}(\mathcal{Q})$$

Lemma 13. *Given a zone-predicate $\mathcal{Q}$ for a $Q \subseteq V_{A_1} \times V_{A_2}$, for any $\mu\nu \in V_{A_1} \times V_{A_2}$, if $\mu\nu \models \texttt{UNot\_sim}_{e_1}^{(C_2^1,\infty)}(\mathcal{Q})$, then $\text{not\_sim}_{e_1}^{(C_2^1,\infty)}(\mu,\nu,Q)$ is true.*
Proof sketch: The sketch is exactly the same as the one for lemma 8. ■

Lemma 13 implies that we may use $\texttt{UNot\_sim}_{e_1}^{(C_2^1,\infty)}(\mathcal{Q})$ to speed up the fixpoint calculation without hurting the precision of the algorithm. With the lemma, we can present our algorithm II, Sim_Check(A_1, A_2), for simulation-checking. It is identical to Sim_Check$^{[0,C_2^1]}(A_1, A_2)$ except that we substitute the following statement (A') for statement (A).

$$\mathcal{Q} := \mathcal{Q} \wedge \left(\neg\texttt{Not\_sim}_{e_1}^{[0,C_2^1]}(Q)\right) \wedge \left(\neg\texttt{UNot\_sim}_{e_1}^{(C_2^1,\infty)}(Q)\right); \ \ldots\ldots\ldots \text{(A')}$$

In procedure $\texttt{OMatch}_{e_1}^{(C_2^1,\infty)}(\mathcal{Q})$, at statement (E), the final value of Π is exactly $\texttt{Match}_{e_1}^{[0,C_2^1]}(\mathcal{Q})$. Thus we can share the evaluation of Π while evaluating $\texttt{Not\_sim}_{e_1}^{[0,C_2^1]}(Q)$ and $\texttt{UNot\_sim}_{e_1}^{(C_2^1,\infty)}(Q)$ in statement (A').

6.4 Complexity of Our Algorithm

The complexity of our algorithm relies on the implementation of the basic manipulation procedures of zones. Like in [11], we argue that we can implement the zones as sets of regions [1]. In such an implementation, basic operations like subsumption, intersection, union, complement, time progression, and variable quantification can all be done in EXPTIME.

Lemma 14. *Proper implementations of algorithms I and II can run in EXPTIME.* ■

7 Implementation and Experiments

We want to check how our techniques work in practice. Moreover, we also want to see how the new formulation in lemma 2 performs in the experiment and how the two algorithms compare with each other. We have implemented the techniques discussed in this manuscript in **RED** 7.0, a model-checker for TAs and parametric safety analysis for LHAs based on CRD and HRD-technology [15,16]. The state-spaces are explored in a symbolic on-the-fly style.

The TAs to **RED** are described as a set of communicating TAs, each representing a process. Several *process transitions* can be combined through CSP-style synchronization events to form a *global transition* [17]. In our implementation, users can specify a particular process as A_1 and another as A_2 and check whether A_1 is simulated by A_2 in the environment of the remaining processes. Thus the simulation checked by **RED** is an extension of the traditional one. This extension seems natural for real-world applications since embedded systems usually have to be verified with respect to certain environments.

Now, a global transition for A_1 is a pair $\langle e_1, e_0\rangle$ such that e_1 is a process transition of A_1 and e_0 is a process transition from the environment processes.

Similarly, a global transition for A_2 is also a pair $\langle e_2, e_0 \rangle$ such that e_2 is a process transition of A_2 and e_0 is a process transition from the environment processes. The simulation checked by **RED** only differs from the traditional one in the definition of transition compatibility between A_1 and A_2. That is, if $\langle e_1, e_0 \rangle$ and $\langle e_2, e_0' \rangle$ are compatible, then not only the event labels and writing values to global variables of e_1 and e_2 must be the same, but also $e_0 = e_0'$. However, this simulation extension is actually reducible to the traditional simulation. The reduction is to assign a unique event symbol to each transition of the environment. Then we compare the combined event symbol of e_1 and e_0 with that of e_2 and e_0 for the compatibility of transitions for A_1 and A_2. This simulation extension can sometimes be checked more efficiently than the traditional one since we do not have to duplicate the environment process states in representing the simulation.

To our knowledge, there is no other tool that supports fully automatic simulation checking for TAs as ours. So we only experimented with our two algorithms. We used the following three parameterized benchmarks from the literature.

1. *Fischer's timed mutual exclusion algorithm* [15]: The algorithm relies on a global lock and a local clock per process to control access to the critical section. Two timing constants used are 10 and 19.
2. *CSMA/CD* [19]: This is the Ethernet bus arbitration protocol with collision-and-retry. The timing constants used are 26, 52, and 808.
3. *Timed consumer/producer*: There is a buffer, some producers, and some consumers. The producers periodically write data to the buffer if it is empty. The consumers periodically wipe out data, if any, in the buffer. The timing constants used are 5, 10, 15, and 20.

For each benchmark, two versions are used, one with a simulation and one without. For the versions with a simulation, A_1 and A_2 are identical. For the version without, A_1 and A_2 differ in only one process transition or invariance condition. For example, for the Fischer's benchmark, the difference is that the triggering condition of a transition to the critical section of A_2 is mistaken. The performance data is reported in table 1. The CPU time used, the total memory consumption for the data-structures in state-space representations, and the total number of fixpoint iterations are reported. As can be seen, the performance of algorithms I and II are roughly the same. Especially, algorithm II did not use fewer iterations to calculate the fixpoints. This could implies that our new formulation of simulation between TAs might actually not hurt the performance in practice.

We have also specifically designed the following fourth benchmark to investigate in what situations, the speed-up technique of algorithm II may prevail.

4. *Periodical samplers*: Two sensors periodically send in sampling signals with periods p_1 and p_2 respectively. When the sampling signals from the two processes happen at the same instant, the operating system needs to do some extra work and may disrupt the service.

The benchmark was designed so that the initial state-pairs can only be rejected through a pre-matching segment of $p_1 \cdot p_2$ time units. As can be seen from the table, for the version with a simulation, algorithm II always uses only 3

Table 1. Performance data of scalability w.r.t. various strategies

benchmarks	versions	m	Algorithm I			Algorithm II		
			time	memory	iter'n	time	memory	iter'n
Fischer's mutual exclusion (m processes)	Simulation exists.	4	31.6s	347k	8	31.3s	320k	8
		5	98.3s	731k	8	92.3s	664k	8
		6	304s	1518k	8	281s	1319k	8
	No simulation exists.	4	9.87s	251k	3	11.7s	250k	3
		5	31.5s	475k	3	28.0s	475k	3
		6	86.0s	957k	3	86.7s	955k	3
CSMA/CD (1 bus+ m senders)	Simulation exists.	1	0.099s	41k	2	0.098s	41k	2
		2	0.80s	177k	2	0.80s	177k	2
		3	122s	3509k	7	125s	3503k	7
	No simulation exists.	1	0.103s	41k	2	0.085s	41k	2
		2	1.50s	202k	2	2.03s	203k	2
		3	21.7s	2089k	2	25.7s	2089k	2
Consumer & producer (1 buffer +1 producer +m consumers)	Simulation exists.	3	0.26s	57k	2	0.30s	57k	2
		4	0.79s	65k	2	0.43s	65k	2
		5	1.21s	76k	2	0.53s	75k	2
	No simulation exists.	3	0.65s	70k	4	0.99s	70k	4
		4	0.93s	77k	4	1.35s	775k	4
		5	1.17s	83k	4	1.16s	83k	4
Periodical sampler (1 process with periods p_1 and p_2)	Simulation exists.	11/13	22.0s	4670k	14	5.86s	2451k	3
		13/17	56.9s	5989k	16	12.7s	3192k	3
		17/19	125s	17846k	20	22.4s	8444k	3
	No simulation exists.	11/13	21.1s	4664k	12	5.43s	2452k	1
		13/17	55.2s	5978k	14	13.0s	3194k	1
		17/19	125s	17842k	18	20.5s	8448k	1

Data collected on a Pentium 4 1.7GHz with 380MB memory running LINUX; s: seconds; k: kilobytes of memory in data-structure; iter'n: the number of iterations.

iterations to reach the fixpoints. On the other hand, the numbers of iterations of algorithm I roughly grow with the values of the periods. The same pattern can be observed in the data for the version without a simulation. As a result, algorithm I runs exponentially slower than algorithm II. In general, the verification tasks for multi-process systems with several exact periods with large least common multiple could incur complexity difference between algorithms I and II.

8 Concluding Remarks

In this work, we discuss how to implement a symbolic simulation-checker for TAs with zones. Our implementation and experiment shows the promise that our algorithm could be useful in practice in the future. A straightforward adaptation to our techniques can be used to check bisimulation (equivalence) between TAs.

References

1. Alur, R., Courcoubetis, C., Dill, D.L.: Model Checking for Real-Time Systems. In: IEEE LICS (1990)
2. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid Automata: an Algorithmic Approach to the Specification and Verification of Hybrid Systems. In: Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.) Hybrid Systems. LNCS, vol. 736, Springer, Heidelberg (1993)
3. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)
4. Aceto, L., Ingólfsdóttir, A., Pedersen, M.L., Poulsen, J.: Characteristic Formulae for Timed Automata. Theoretical Informatics and Applications 34(6), 565–584 (2000)
5. Bengtsson, J., Griffioen, W.O.D., Kristoffersen, K.J., Larsen, K.G., Larsson, F., Pettersson, P., Wang, Y.: Verification of an Audio Protocol with Bus Collision Using UPPAAL. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, Springer, Heidelberg (1996)
6. Beyer, D.: Efficient Reachability Analysis and Refinement Checking of Timed Automata Using BDDs. In: Margaria, T., Melham, T.F. (eds.) CHARME 2001. LNCS, vol. 2144, Springer, Heidelberg (2001)
7. Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Wang, Y.: UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In: Hybrid Control System Symposium. LNCS, Springer, Heidelberg (1996)
8. Cerans, K.: Decidability of bisimulation equivalence for parallel timer processes. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, Springer, Heidelberg (1993)
9. Cleaveland, R., Steffen, B.: Computing behavioral relations, logically. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) Automata, Languages and Programming. LNCS, vol. 510, pp. 127–138. Springer, Heidelberg (1991)
10. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: IEEE FOCS'95 (1995)
11. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic Model Checking for Real-Time Systems. In: IEEE LICS (1992)
12. Lin, H., Wang, Y.: Axiomatising timed automaton. Acta Informatica 38(4), 277–305 (2002)
13. Nakata, A.: Symbolic Bisimulation Checking and Decomposition of Real-Time Service Specifications. Ph.D. Dissertation, Faculty of the Engineering Science, Osaka University (1997)
14. TaşIran, S., Alur, R., Kurshan, R.P., Brayton, R.K.: Verifying abstractions of timed systems. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, Springer, Heidelberg (1996)
15. Wang, F.: Efficient Verification of Timed Automata with BDD-like Data-Structures, STTT (Software Tools for Technology Transfer), 6(1) (2004) Springer-Verlag; special issue for the 4th VMCAI, January 2003, LNCS 2575, Springer-Verlag
16. Wang, F.: Symbolic Parametric Safety Analysis of Linear Hybrid Systems with BDD-like Data-Structures. IEEE Transactions on Software Engineering, 31(1), 38–51 (2005) IEEE Computer Society. A preliminary version is in proceedings of 16th CAV, 2004, LNCS 3114, Springer-Verlag

17. Wang, F.: Symbolic Verification of Distributed Real-Time Systems with Complex Synchronizations. In: Lau, K.-K., Banach, R. (eds.) ICFEM 2005. LNCS, vol. 3785, Springer, Heidelberg (2005)
18. Wang, F., Huang, G.-D., Yu, F.: TCTL Inevitability Analysis of Dense-Time Systems: From Theory to Engineering. IEEE Transactions on Software Engineering 32(7) (2006)
19. Kronos, S.Y.: A Verification Tool for Real-Time Systems. International Journal of Software Tools for Technology Transfer 1(1/2) (1997)

Author Index